

CUBIT Mesh Generation Environment Volume 1: Users Manual

Cubit Development Team¹
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

The CUBIT mesh generation environment is a two- and three-dimensional finite element mesh generation tool which is being developed to pursue the goal of robust and unattended mesh generation—effectively automating the generation of quadrilateral and hexahedral elements. It is a solid-modeler based pre-processor that meshes volume and surface solid models for finite element analysis. A combination of techniques including paving, mapping, sweeping, and various other algorithms being developed are available for discretizing the geometry into a finite element mesh. CUBIT also features boundary layer meshing specifically designed for fluid flow problems. Boundary conditions can be applied to the mesh through the geometry and appropriate files for analysis generated. CUBIT is specifically designed to reduce the time required to create all-quadrilateral and all-hexahedral meshes. This manual is designed to serve as a reference and guide to creating finite element models in the CUBIT environment.

| This manual documents CUBIT Version 1.14.

1. See the next page for the members of the CUBIT Development Team.

▼ Cubit Development Team Membership

Sandia National Laboratories, Albuquerque New Mexico

Patrick Knupp	Parallel Computing Sciences
Randy R. Lober	Advanced Engineering & Manufacturing Software
Darryl Melander	Parallel Computing Sciences
Scott A. Mitchell	Parallel Computing Sciences
Gregory D. Sjaardema	Solid & Structural Mechanics
W. Ann Sample	Parallel Computing Sciences
Marilyn K. Smith	Technology Programs
Timothy J. Tautges	Parallel Computing Sciences
David R. White	Parallel Computing Sciences

Brigham Young University, Provo, Utah

Steve Benzley	Professor of Civil and Environmental Engineering
Robert Kerr	Student in Department of Civil and Environmental Eng.
Steven R. Jankovich	Student in Department of Mechanical Engineering
Daniel B. McRae	Student in Department of Civil and Environmental Eng.
Jason Shephard	Student in Department of Civil and Environmental Eng.

University of Wisconsin

Contractors

Ray J. Meyers	Contractor, Provo, Utah
---------------	-------------------------

▼ Table of Contents

▼ Cubit Development Team Membership	4
▼ Table of Contents	5
▼ List of Figures	13
▼ List of Tables	15
 Chapter 1: Getting Started	 17
▼ How to Use This Manual	17
▼ CUBIT Mailing List	18
▼ Problem Reports and Enhancement Requests	18
▼ Executing CUBIT	19
Execution Command Syntax	19
Initialization File	20
User Environment Settings	20
Graphics Customization	21
▼ Command Syntax	21
Identifier Ranges	23
▼ Features	23
Geometry Creation	24
Algebraic Command Preprocessing (APREPRO)	24
Geometry Consolidation	24
Geometry Decomposition	24
Supported Element Types	24
Mesh Creation	24
Boundary Condition Application	25
Graphical Display Capabilities	25
Hardware Platforms	25
▼ Future Releases	25
 Chapter 2: Tutorial	 27
▼ The Tutorial	27
▼ Step 1: Beginning Execution	29
▼ Step 2: Creating the Brick	29
▼ Step 3: Creating the Cylinder	31
▼ Step 4: Adjusting the Graphics Display	31
▼ Step 5: Forming the Hole	32
▼ Step 6: Setting Body Interval Size	33
▼ Step 7: Setting Specific Surface Intervals	33
▼ Step 8: Setting Specific Curve Intervals	34
▼ Step 9: Surface Meshing	35
▼ Step 10: Volume Meshing	35

▼ Congratulations!	37
Chapter 3: Environment	39
▼ Interface Choices	39
Command Entry	39
No-Graphics Interface.	40
Batch Execution	41
▼ Session Control	41
▼ Command Journalling	41
CUBIT Journal File Generation	41
Replaying Journal Files.	42
▼ Graphics	42
Graphics Window Control	42
Image Rendering Control	43
Viewing the Model	45
Displaying Entities	48
Drawing Entities 48	
Highlighting Entities 48	
Setting Visibility 49	
Global Settings	49
Individual Geometric Entity Settings	49
Color	50
Entity Labeling	50
Hardcopy Output.	51
Video Animations.	51
▼ Model Information	52
Model Summary Information	52
Geometry Information	52
Mesh Information	54
Special Entity Information	56
Other Information	57
Message Output Settings 58	
Graphical Display Information 59	
Memory Usage Information 59	
▼ Picking	60
▼ Help Facility	61
Chapter 4: Geometry	63
▼ Geometry Definition	63
Topology.	63
Vertex 64	
Curve 64	
Surface 64	

Volume 64	
Body 64	
Group 64	
Groups.....	64
Geometry Entity Identifier Ranges.....	65
Cellular Topology.....	65
▼ Geometry Creation	66
Geometric Primitives	66
Brick 66	
Cylinder 67	
Prism 68	
Frustum 68	
Pyramid 68	
Sphere 69	
Torus 69	
Importing Geometry	69
Importing ACIS Models 70	
Importing FASTQ Models 70	
Importing EXODUSII Files 70	
Importing PRO/Engineer Models 71	
Exporting Geometry	71
▼ Geometry Manipulation	71
Transform Operations.....	72
Copy 72	
Move 72	
Scale 72	
Rotate 72	
Reflect 73	
Restore 73	
Boolean Operations.....	73
Intersect 73	
Subtract 73	
Unite 73	
Sweep Operations	74
Imprint Operation	75
▼ Geometry Decomposition	75
Web Cutting	75
Body-Based Decomposition.....	76
▼ Geometry Consolidation	76
General Geometry Consolidation	78
Selective Geometry Consolidation	78
▼ Geometry Attributes	79
Entity Names.....	79
 Chapter 5: Mesh Generation	 81

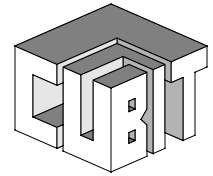
▼ Mesh Definition	81
Mesh Hierarchy	81
Node 81	
Edge 82	
Face 82	
Hex 82	
Mesh Generation Order of Procession	82
▼ Mesh Attributes	82
Meshing Schemes	82
Mesh Density Specification	86
Element Types	87
▼ Curve Meshing	87
Interval Firmness and Parity	88
Standard Node Density	88
Relative Element Edge Lengths	88
Sizing Function-Based Node Density	89
Featuresize Function Node Density	89
Meshing the Curve	90
▼ Surface Meshing	91
Surface Vertex Types	91
Mapping and Submapping Interval Constraints	91
Scheme Designation	94
Surface Mapping 94	
Paving 94	
Surface Submapping 95	
Meshing Primitives 96	
Adaptive Surface Meshing	97
Boundary Layer Meshing	103
Generating the Surface Mesh	105
▼ Volume Meshing	105
Scheme Designation	105
Volume Mapping 106	
Volume Submapping 107	
Sweeping (Project, Translate, and Rotate) 108	
Project	109
Translate	111
Rotate	111
Plastering 111	
Stair Tool 112	
Whisker weaving 113	
Whisker weaving basic commands 114	
Viewing the Weave 114	
Resolving whisker weaving degeneracies 115	
Knife resolution	115
Doublet Resolution	117
Degeneracy Resolution Using Pillow Sheets	117

Weaving without geometry	118
Miscellaneous Whisker Weaving Options	119
Whisker Weaving Sub-Command Interface	119
Explicit Looping	120
Deleting loop self-intersections	120
Weaving database while plastering	121
Dicing	121
Dicer Sheets and Refinement Intervals	122
Dicing Basic Commands	122
Additional Dicing Commands	123
Generating the Volume Mesh	124
▼ Mesh Editing	124
Mesh Smoothing	124
Surface smoothing	124
Volume smoothing	125
Mesh Deletion	126
Complete mesh removal	126
Partial mesh removal	126
Individual mesh face removal	127
Node and NodeSet Repositioning	127
▼ Mesh Importing and Duplicating	127
Importing mesh from an external file	127
Duplicating mesh	128
▼ Mesh Quality	128
Background	128
Command Syntax	129
Command Examples	130
Example Output	130
 Chapter 6: OvFinite Element Model Definition and Output	 133
▼ Finite Element Model Definition	133
Element Blocks	133
Nodesets	133
Sidesets	134
Property Names	134
▼ Element Block Specification	134
Default Element Types, Block IDs, and Attributes	135
Element Block Definition Examples	135
Multiple Element Blocks	135
Surface Mesh Only	135
Two-Dimensional Mesh	135
▼ Boundary Conditions: Nodesets and Sidesets	136
Nodeset Associativity Data	136
▼ Setting the Title	137

Table of Contents

▼ Exporting the Finite Element Model	137
Appendix A: Command Index	139
▼ Command Syntax	139
▼ Commands	139
Appendix B: Examples	159
▼ General Comments	159
▼ Simple Internal Geometry Generation	160
▼ Octant of Sphere	161
▼ Airfoil	163
▼ The Box Beam	164
▼ Thunderbird 3D Shell	167
▼ Assembly Components	170
▼ Whisker Weaving	174
Appendix C: CUBIT Installation	175
▼ Licensing	175
▼ Distribution Contents	176
▼ Installation	176
▼ HyperHelp Installation	176
System Requirements	177
CPU	177
Disk Space	177
Printer	177
Operating System	177
Windowing Environment	178
Copying HyperHelp Files	178
Setting Up the HyperHelp Environment	178
Appendix D: Available Colors	181
References	185
Glossary	187
Index	191

Table of Contents

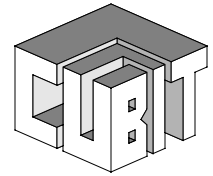


▼ List of Figures

Figure 2-1	Geometry for Cube with Cylindrical Hole.....	28
Figure 2-2	Generated Mesh for Cube with Cylindrical Hole	28
Figure 3-1	Schematic of From, At, Up, and Perspective Angle	45
Figure 4-1	Cellular Topology Between Volumes.....	65
Figure 4-2	Dangling Faces & Edges.....	65
Figure 4-3	CUBIT Geometry Primitives	67
Figure 4-4	Solid Model Prior to Decomposition	77
Figure 4-5	Solid Model After Decomposition.....	78
Figure 5-1	Model Meshed Using Automatic Scheme Selection	85
Figure 5-2	Local Node Numbering for CUBIT Element Types.....	87
Figure 5-3	Curves meshed with featuresize.....	90
Figure 5-4	Equal and biased curve meshing.....	90
Figure 5-5	Illustration of Angle Types	92
Figure 5-6	Scheme Map Logical Properties	92
Figure 5-7	Scheme Submap Logical Properties	93
Figure 5-8	Mapped and paved surface meshing	94
Figure 5-9	Submapping Example	96
Figure 5-10	Alternate Submapping Topology Interpretation	96
Figure 5-12	Triangle and Circle Primitive Meshes	97
Figure 5-11	Periodic Surface Meshing with Submapping.....	97
Figure 5-13	NURB solid with high surface curvature change	98
Figure 5-14	NURB mesh with no interior sizing function	99
Figure 5-15	NURB mesh with curvature sizing function.....	99
Figure 5-16	NURB mesh with no sizing function, 34 by 16 density	99
Figure 5-17	NURB mesh with linear sizing function, 34 by 16 density	100
Figure 5-18	NURB mesh with interval sizing function, 34 by 16 density	100
Figure 5-19	NURB mesh with inverse sizing function, 34 by 16 density.....	101
Figure 5-20	NURB mesh with super sizing function, 34 by 16 density.....	101
Figure 5-21	Test sizing function mesh for square geometry	102
Figure 5-22	Test sizing function for spline geometry	102
Figure 5-23	Plastic strain metric and the adaptively generated mesh	104
Figure 5-24	Boundary Layer Parameters.....	104
Figure 5-25	Volume mapping of an 8-surfaced volume.....	106
Figure 5-26	Volume mapping of a 5-surfaced volume.....	107
Figure 5-27	Surface mesh of an 8-surfaced volume highlighting the logical edges used for volume mapping.	107
Figure 5-28	Example of internal virtual surface creation.....	108
Figure 5-29	Hexahedral mesh generated by volume submapping.....	108
Figure 5-30	Project Volume Meshing	109
Figure 5-31	Multiple Surface Project Volume Meshing	110
Figure 5-32	Plastering Examples.....	112

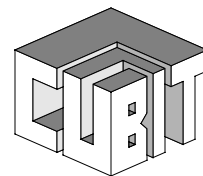
List of Figures

Figure 5-33	StairTool mesh.	113
Figure 5-34	Whisker weaving meshes.	113
Figure 5-35	Example sheet diagram (left) and corresponding loop (right).	115
Figure 5-36	Illustration of Quadrilateral Shape Parameters (Quality Metrics)	129
Figure 5-37	Illustration of Quality Metric Graphical Output	131



▼ List of Tables

Table 3-1	Command Line Interface Line Editing Keys	40
Table 3-1	CUBIT Journal file used for List Output Examples	52
Table 3-2	Sample Output from 'List Model' Command	53
Table 3-3	Sample Output from 'List Names' Command	53
Table 3-4	Sample Output from 'List Surface Ids' Command	54
Table 3-5	Sample Output from 'List Group' Command	54
Table 3-6	Sample Output from 'List Body' Command	54
Table 3-7	Sample Output from 'List Volume' Command	55
Table 3-8	Sample Output from 'List Surface' Command	55
Table 3-9	Sample Output from 'List Curve' Command	56
Table 3-10	Sample Output from 'List Vertex' Command	56
Table 3-11	Sample Output from 'List Hex' Command	56
Table 3-12	Sample Output from 'List Face' Command	57
Table 3-13	Sample Output from 'List Edge' Command	57
Table 3-14	Sample Output from 'List Node' Command	57
Table 3-15	Sample Output from 'List Block' Command	57
Table 3-16	Sample Output from 'List SideSet' Command	58
Table 3-17	Sample Output from 'List NodeSet' Command	58
Table 3-18	Sample Output from 'List Settings' Command	59
Table 3-19	Sample Output from 'List View' Command	60
Table 3-20	Sample Output from 'List Memory' Command	60
Table 5-1	Default Meshing Attributes	82
Table 5-2	Valid Meshing Schemes for Curves, Surfaces, and Volumes	83
Table 5-1	Listing of logical sides	93
Table 5-1	. Whisker weaving sub-command line interface commands.	119
Table 5-1	Sample Output for 'Quality' Command	130
Table 5-1	Element Quality Plot Legend	130
Table 6-1	.Nodeset id base numbers for geometric entities	137
Table B-1	CUBIT Features Exercised by Examples.	160
Table C-1	HyperHelp Distribution Files	177
Table 6-2	Available Colors	181



Chapter 1: Getting Started

- ▼ How to Use This Manual...17
- ▼ CUBIT Mailing List...18
- ▼ Problem Reports and Enhancement Requests...18
 - ▼ Executing CUBIT...19
 - ▼ Command Syntax...21
 - ▼ Features...23
 - ▼ Future Releases...25

Welcome to CUBIT, the Sandia National Laboratory automated mesh generation toolkit. With CUBIT the geometry of a part can be imported, created, and/or modified using an embedded solid modelling engine. The geometry can then be discretized into a finite element mesh using a combination of techniques including paving [1], mapping, sweeping, and various other algorithms being developed. Boundary conditions can be applied to the mesh through the geometry and appropriate files for analysis generated. CUBIT is specifically designed to reduce the time required to create all-quadrilateral and all-hexahedral meshes.

▼ How to Use This Manual

This manual provides specific information about the commands and features of CUBIT. It is divided into chapters which roughly follow the process in which a finite element model is designed, from geometry creation to mesh generation to boundary condition application. An example is provided in a tutorial chapter to illustrate some of the capabilities and uses of CUBIT. Appendices containing complete command usage, examples, installation instructions, and a list of available colors are included.

The CUBIT environment is designed to provide the user with powerful meshing algorithms that require minimal input to produce a complete finite element model. As such, the code is constantly being updated and improved. Feedback from our users indicates that new meshing tools are often needed and/or desired before they have been completely tested and debugged. As a service to the user, these tools are integrated and made available as quickly as possible, but in a *user beware* state. As they are further tested (often with the assistance of users) and improved, the state of the particular tool becomes more stable, and thus the risk to the user is lowered. Since documentation of the tool is necessary for actual use, we have included the documentation



of all available tools in the manual. However, to warn the user, a “hammer” icon is placed in the document next to those features that are only minimally tested or are in a state of work-in-progress (See “hammer” icon in left margin). In other words, “proceed with caution.” Certain portions of this manual contain information that is vital for understanding and effectively using CUBIT. In order to highlight these portions, a “key” icon is positioned in the document next to these sections. In other words, “this is a key point”.

This manual is Volume 1 of the CUBIT documentation set. The companion document is *CUBIT Mesh Generation Environment, Volume 2: Developers Manual* [3] which contains internal programming-related details of the CUBIT mesh generation environment.

This manual documents CUBIT Version 1.14, December 15, 1996.

▼ CUBIT Mailing List

A mailing list has been created to keep interested users informed of new features, bug-fixes, and other pertinent information about CUBIT. The list can also be used for general discussions about CUBIT. Users can subscribe to the mailing list by sending a mail message to **listserv@sahp046.jal.sandia.gov** with the body (not the subject) of the mail message containing the line:

subscribe cubit Your Full Name

The user would then receive a message confirming the subscription to the CUBIT mailing list. More information about the use of the mailing lists can be obtained by sending the message **help** to the above mail address. Messages are sent to the list by sending mail to the address:

cubit@sandia.gov

The CUBIT developers will be sending announcements of new CUBIT capabilities, enhancements, and user-visible bug fixes to this list on a regular basis. In addition, this list may be used for general questions regarding CUBIT that may be solvable by other subscribers to the list.

An additional mailing list has been created for direct communication with the CUBIT developers. All messages sent to this list will be distributed to the CUBIT developers only. It should be used for questions that are not of general interest to other CUBIT users. Messages are sent to the CUBIT developers by sending mail to the address:

cubit-dev@sandia.gov

▼ Problem Reports and Enhancement Requests

All problem reports and enhancement requests for CUBIT should be sent to **cubit-dev@sandia.gov**. These requests will be responded to as quickly as possible. As the number of CUBIT users increases, it may be beneficial in the future to go to an electronic bug reporting system; instructions for this system will be e-mailed to **cubit@sandia.gov** when appropriate.



Note: The existence and recommended use of an electronic mailing list to report bugs and request enhancements is not an attempt by the CUBIT developers to ignore and or

discourage face-to-face discussion of problems with, or enhancements to the CUBIT code with users.

▼ Executing CUBIT

Execution Command Syntax

Two versions of CUBIT are currently supported¹: 1) a basic command line version which outputs graphics to a standard X Window System graphics window, and 2) a command line version with no graphics. Both versions can be run in “batch” mode, requiring no direct interaction from the user. The commands to execute these versions of CUBIT on most systems are simply:

cubit Command line version with X Window system graphics.

cubitb Command line version with no graphics.

Throughout this manual, “CUBIT” will be used as a generic term that applies to all of the executables. If it is necessary to specify a specific version of CUBIT, one of above names will be used.

The command syntax recognized by CUBIT is:

```
{cubit|cubitb} [-help] [-initfile <val>] [-noinitfile] [-solidmodel <val>]
[-batch] [-nojournal] [-journalfile <file>] [-maxjournal <val>]
[-noecho] [-debug=<val>] [-information={on|off}] [-warning={on|off}]
[-Include <path>] [-fastq <fastq_file>] {<input_file_list>|<var=value>}...
```

where the quantities in square brackets **[-options]** are optional parameters that are used to modify the default behavior of CUBIT and the quantities in angle brackets **<values>** are values supplied to the option. The effect of these parameters are:

-help Print a short usage summary of the command syntax to the terminal and exit.

-initfile <val> Use the file specified by **<val>** as the initialization file instead of the default initialization file **\$HOME/.cubit**.

-noinitfile Do not read any initialization file. The default behavior is to read the initialization file **\$HOME/.cubit** or the file specified by the **-initfile** option if it exists.

-solidmodel <val> Read the ACIS solid model geometry information from the file specified by **<val>** prior to prompting for interactive input.

-batch Specify that there will be no interactive input in this execution of CUBIT. CUBIT will terminate after reading the initialization file, the geometry file, and the file specified by the **-initfile** option.

-nojournal Do not create a journal file for this execution of CUBIT. This option performs the same function as the **Journal Off** command. The default behavior is to create a journal file.

-journalfile <file> Write the journal entries to the file **<file>**. The file will be overwritten if it already exists.

1. In subsequent versions of CUBIT, a single executable will be used for both graphics and no-graphics versions; graphics behavior will be controlled through a command line option and/or a CUBIT command.

-maxjournal <val> Only create a maximum of **<val>** default journal files. Default journal files are of the form **cubit.#.jou** where # is a number in the range 01 to 99.

-noecho Do not echo commands to the console. This option performs the same function as the **Echo Off** command. The default behavior is to echo commands to the console.

-debug=<val> Set the debug message flags indicated by **<val>**. **<val>** is a comma-separated list of integers or ranges of integers. An integer range is specified by separating the beginning and the end of the range by a hyphen. For example, to set debug flags 1, 3, and 8 to 10 on, the syntax would be **-debug=1,3,8-10**. Flags not specified are off by default. Debug messages are typically of importance only to developers and are not normally used in normal execution.

-information={on|off} Turn on/off the printing of information messages from CUBIT to the console.

-warning={on|off} Turn on/off the printing of warning messages from CUBIT to the console.

-include=<include_path> Set the patch to search for journal files and other input files to be **<include_path>**. This is useful if you are executing a journal file from another directory and that journal file includes other files that exist in that directory also.

-fastq=<fastq_file> Read the mesh and geometry definition data in the FASTQ file **<fastq_file>** and interpret the data as FASTQ commands. See Reference [5] for a description of the FASTQ file format.

The information following the last option on the command line consists of either input files or variable definitions. Variables are specified by the syntax **<variable=value>** where variable is any valid Aprepro variable name (See Reference [13]) and value is either a real value or a string value. String values must be surrounded by double quotes. Input files are specified simply by typing the filename. All files specified on the command line following the last option are processed in the order they are listed prior to prompting for interactive command input.

An example of the use of the command line options is:

```
cubitb -batch -nojournal final_mesh.jou height=1.2345
```

which specifies that **cubitb** will execute the commands in the file **final_mesh.jou** unattended. The Aprepro variable **height** will be defined to have the value 1.2345. This mode is typically used to recreate a previously generated mesh with no user interaction.

The command options can also be specified through the **CUBIT_OPT** environment variable. See the “User Environment Settings” section below for more information.

Initialization File

If the file **\$HOME/.cubit** or the file specified by the optional **-initfile <val>** option exists when CUBIT begins executing, it is read prior to beginning interactive command input. This file is typically used to perform initialization commands that do not change from one execution to the next, such as turning off journal file output, setting geometric and mesh entity colors, and setting the size of the graphics window.

User Environment Settings

To execute CUBIT several environment variables must be set. In particular the “**HOME**”, “**PATH**” and “**DISPLAY**” variables. The **HOME** environment variable is typically set automatically when you login to a system. Its purpose is to provide a pointer to your login

directory. The **PATH**, on a Unix system, is a list of directories that are searched for commands to be executed. Proper setting of the path is system-dependent; if CUBIT does not execute correctly, contact your system manager or another CUBIT user for the correct setting of the **PATH** specification.

The X Window System-based command line input version of CUBIT (**cubit**) requires the specification of the **DISPLAY** environment variable which is used by the application to determine where the graphics window should be displayed (and which screen should be used on displays with multiple monitors).

CUBIT can also read the environment variable **CUBIT_HELP_DIR** for use with the online hypertext help system. This variable should be set to the pathname specifying where the CUBIT help file **cubitHelp.hlp** is located. The person responsible for installing CUBIT on the system should be contacted for this information.

Another useful environment variable is **CUBIT_OPT** which can be used to set execution command line parameter options. For example, if journalling of commands is never wanted, then setting **CUBIT_OPT** to **-nojournal** will turn off journalling for all CUBIT executions¹.

Graphics Customization

Settings for the default CUBIT window sizes, locations, colors, and fonts can be set in the **.Xdefaults** or **.Xresources** resource files in the user's home directory. This file is a text file that can be edited with any standard UNIX text editor. In the resource file, each resource must be on a separate line. The resource setting consists of a resource label, a colon, one or more spaces or tabs, and the resource value. For additional general information on resources, see X Window System documentation; a readily available documentation source is the **RESOURCES** section of the X(1) manual page which is usually installed on most systems.

A CUBIT resource label begins with the word "cubit" followed by an asterisk or period, followed by the specification of the resource. For example, to specify that the CUBIT graphics window should be 700 pixels square rather than the default size, the following line should be added to the resource file:

```
cubit*CUBIT.geometry: 700x700+445+0
```

For example:

```
cubit*XmTextField*background: LightBlue
```

This line states that the background of all text fields in the **cubit** application will be the color LightBlue. Colors can normally be found in the `/usr/lib/X11/rgb.txt` file on your system. For additional information about **Xdefaults** or **Xresources** files, see the application defaults section in any X Window user's book.

▼ Command Syntax

The execution of CUBIT is controlled either by entering commands from the command line or by reading them in from a journal file. Throughout this document, each function or process will have a description of the command required to perform the function or process. In this section, the command syntax used in this manual will be described. The user can obtain a quick guide

1. Journalling could then be turned back on with the "Record" command.

to proper command format by issuing the **<keyword> help** command. This help message will indicate the full command syntax expected by the keywords. For example, entering **view help** results in the following output:

```
View At <X_coord> <Y_coord> <Z_coord>
View From <X_coord> <Y_coord> <Z_coord>
View List
View Up <X_coord> <Y_coord> <Z_coord>
```

The words that begin with an uppercase letter are keywords which must be entered (case is not significant) and the bracketed words are user supplied parameters.

The commands recognized by CUBIT are free-format and must adhere to the following syntax rules.

- Either lowercase or uppercase letters are acceptable.
- The “#” character in any command line begins a comment. The “#” and any characters following it on the same line are ignored.

Each command typically has either:

- an action keyword or “verb” followed by a variable number of parameters, for example

Mesh Volume 1

- or a selector keyword or “noun” followed by a “verb” or “selector keyword and a variable number of parameters, for example

Volume 1 Scheme Project Source 1 Target 2

The action or selector keyword is a character string matching one of the valid commands. It may be abbreviated as long as enough characters are used to distinguish it from other commands. The meaning and type of the parameters depend on the keyword. Valid entries for parameters are:

- A numeric parameter may be a real number or an integer. A real number may be in any legal C or FORTRAN numeric format (for example, 1, 0.2, -1e-2). An integer parameter may be in any legal decimal integer format (for example, 1, 100, 1000, but not 1.5, 1.0, 0x1F).
- A string parameter is a literal character string contained within single or double quotes. For example, **‘This is a string’**.
- A filename parameter must specify a legal filename on the system that CUBIT is running. Environment variables and aliases may not be used in the filename specification. For example, the C-Shell shorthand of referring to a file relative to the user’s login directory (**~jdoe/cubit/mesh.jou**) is not valid. The filename must be specified using either a relative path (**../cubit/mesh.jou**), or a fully-qualified path (**/home/jdoe/cubit/mesh.jou**). Like a string, it also must be contained within single quotes.
- Some commands require a “toggle” keyword to enable or disable a setting or option. Valid toggle keywords are **“on”**, **“yes”**, and **“true”** to enable the option; and **“off”**, **“no”**, and **“false”** to disable the option.

The notation conventions used in the command descriptions in this document are:

- The command will be shown in a format that **looks like this**,
- A word enclosed in angle brackets (**<parameter>**) signifies a user-specified value. The value can be an integer, a range of integers, a real number, or a string. The valid value types should be evident from the command or the command description.

- A series of words in braces and delimited by a vertical bar (**{choice1 | choice2 | choice3}**) signifies that one of the words within the brackets must be entered.
- A word enclosed in square brackets (**[optional]**) signifies an optional parameter which can be entered to modify the default behavior of the command, but is not required.

An example of this command syntax is shown below.

{volume surface curve} <range> size <size>	
volume 1 size 0.5	Valid
surface 1 to 10 by 3 size 0.05	Valid
volume 10 to 1 size 0.05	Invalid — negative increment
volume 10 to 1 by -1 size 0.05	Valid
surface 1 10 size 1.0	Invalid — not a valid range specification
surface 1 to 10 interval 5.0	Invalid — “interval” requires an integer

Identifier Ranges

Integers are used to identify various types of objects in the model. For example, a model might have 3 volumes, numbered 2, 3 and 5. Commands entered by the user often apply to multiple objects; the ids for these objects can be specified using id ranges. The parsing of id ranges in CUBIT is done according to the following syntax:

<id_range> = n1 n2 n3	Selects ids n1, n2 and n3
<id_range> = n1 to n2 by n3	Selects identifiers between n1 and n2 stepping by n3
<id_range> = all	Selects all identifiers for a given object type

Note that the first two varieties described above can be used in sequence to describe an arbitrary identifier range; for example, the following range specification is valid:

<id_range> = n1 n2 n3 to n4 by n5 n7 to n8

In addition to the using general identifier ranges described above, geometric objects can be specified by topological relationship and other criteria; for more information, See “Geometry Entity Identifier Ranges” on page 65.

▼ Features

The CUBIT environment is designed to provide the user with powerful meshing algorithms that require minimal input to produce a complete finite element model. CUBIT is based on a solid modeler that provides it with a precise geometric representation. The *paving* algorithm [1] has been extended to mesh complex three dimensional surfaces based on the solid modeler. Volumetric meshing is provided by mapping transformations and sweeping algorithms. Several quadrilateral and hexahedral element types are also supported. The following sections provide a brief overview of the CUBIT meshing toolkit.

Geometry Creation

Geometry creation is accomplished using the geometric primitives and boolean operations in CUBIT or by reading an external solid model file into the CUBIT meshing toolkit. External solid model files can be created from any of several environments that support the ACIS® solid model format: a rudimentary command line system, referred to as the “test harness [4],” is useful for building quick and straightforward models. Other more advanced environments include the Aries® ConceptStation and PRO/Engineer via a PRO/Engineer/ACIS translator. Models from FASTQ [5] input files can be imported into CUBIT and meshed.

Algebraic Command Preprocessing (APREPRO)

Many analysts use the Aprepro [13] program to preprocess commands and journal files which contain algebraic expressions. The Aprepro algebraic preprocessing capability has been implemented into the CUBIT command parser. The full Aprepro functionality has been included except for the **units**, and **loop** commands.

Geometry Consolidation

When assembly solid models are imported into the CUBIT environment, many surface, curve, and vertex entities will be redundant. To resolve this issue, the automated geometry consolidation or “merge” routines will identify matching entities and make database modifications to remove the redundancy. Geometry consolidation can also be interactive, in case certain redundant features need to be retained to represent slide surfaces or slide lines. The geometry merge capability eliminates the generation of non-contiguous elements between adjacent surfaces and curves which would have to be removed after meshing.

Geometry Decomposition

Solid models imported into the CUBIT environment sometimes consist of combinations of simple geometric volumes, for example a plate with a cylinder projecting out of it. These geometries are also sometimes constructed within CUBIT. Currently these geometries must be decomposed into topologically primitive lumps (cylinder, brick, etc) before being able to be meshed. CUBIT contains functions which aid in the decomposition of complicated geometries into meshable pieces.

Supported Element Types

Element types supported in CUBIT include 2 and 3 node bars and beams; 4, 8, and 9 node quads; 4, 8, and 9 node shells; and 8, 20, and 27 node hex elements. Element types must be set before mesh generation is initiated.

Mesh Creation

Mesh generation in CUBIT is designed to be highly automated although numerous control mechanisms are provided to allow the user to guide the meshing process. Meshing is controlled through scheme choice, and interval number or node density specification. Curve meshing schemes include equally spaced and biased spaced intervals. Surface meshing schemes include mapping transformations, paving, boundary layers, and primitives. Volume meshing schemes

include mapping transformations, mesh sweeping or projecting, multiple surface sweeping, and submapping. In addition, automated volume meshing algorithms are being developed.

Boundary Condition Application

Once a suitable mesh has been generated, elements can be grouped into sets using three control classes: element blocks, nodesets, and sidesets. Numeric flags are associated with these sets allowing analysis codes to apply appropriate boundary conditions to the correct mesh entities.

Element blocks are used for efficient storage of a finite element model. Within an element block, all elements are of the same type (basic geometry and number of nodes) and have the same material definition.

Nodesets provide a means to reference a group of nodes with a single identification number rather than by each node's identification number. Nodesets are typically used to specify load or boundary conditions, or to identify a set of nodes for a special processing within CUBIT. A node may appear in multiple nodesets, but will only appear once in any single nodeset.

Sidesets provide an additional means of applying load and boundary conditions. Unlike nodesets, sidesets group sides or faces of elements rather than simply a list of nodes. For example, a pressure load must be associated with elements rather than nodes to apply it properly.

Nodes, element edges, and element faces can belong to multiple nodesets and sidesets. Nodesets and sidesets can be individually displayed for visual inspection. See reference [6] for more information.

Graphical Display Capabilities

CUBIT uses the Hoops graphic display environment to render images. CUBIT can display a wireframe, hiddenline, or shaded representation of geometric and mesh entities. CUBIT can also generate a PostScript file of any displayed image (see "Hardcopy Output" on page 51). Complete control over the viewport parameters and the zoom magnification provide the user with an intuitive modeling environment. Users can also perform screen picking and point-and-click rotate, pan, and zoom operations.

Hardware Platforms

CUBIT is written in "standard" C++ and should execute on any Unix operating system. To date, it has been compiled and used on Sun (both SunOS and Solaris), Hewlett-Packard (HP-UX 9.X), and Silicon Graphics workstations (IRIX 5.3).

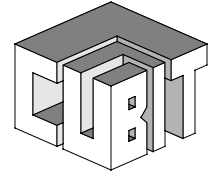
▼ Future Releases

CUBIT is currently on a 6-month major release cycle (April and October); more current versions are also available if required. The capabilities of CUBIT will be expanded and enhanced on a regular basis as dictated by user needs and the developmental progress of new meshing algorithms. Areas of concentration will include

- full-featured automatic hexahedral meshing using a combination of *plastering* and *whisker-weaving*,
- quadrilateral and hexahedral adaptivity,

- enhanced geometric functionality such as overlapping geometry consolidation and more robust geometry decomposition.
- improved control of the naming of geometric and mesh entities.
- improved usability, robustness, and functionality.

Extension of the CUBIT environment to new platforms will also be pursued according to user needs.



Chapter 2: Tutorial

▼ The Tutorial...	27
▼ Step 1: Beginning Execution...	29
▼ Step 2: Creating the Brick...	29
▼ Step 3: Creating the Cylinder...	31
▼ Step 4: Adjusting the Graphics Display...	31
▼ Step 5: Forming the Hole...	32
▼ Step 6: Setting Body Interval Size...	33
▼ Step 7: Setting Specific Surface Intervals...	33
▼ Step 8: Setting Specific Curve Intervals...	34
▼ Step 9: Surface Meshing...	35
▼ Step 10: Volume Meshing...	35
▼ Congratulations!...	37

The purpose of this chapter is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a brief tutorial on the use of the software package. This chapter is designed to demonstrate step-by-step instructions on generating a simple mesh on a perforated block.

▼ The Tutorial

The following is a sample of the basics of using CUBIT to generate and mesh a geometry. By following this tutorial, you will become familiar with the command-line interface and with as much of the CUBIT environment as possible without stopping for detailed explanations. All the commands introduced in this tutorial are thoroughly documented in subsequent chapters. Here are a few tips in following the example in the tutorial

- Focus on instructions preceded with “Step” numbers. These step you through a series of explicit activities that describe exactly what to do to complete the task.
- Refer to screen shots and other pictures that show you what you should see on your own display as you progress through the tutorial.

- An example of the command line is shown below. In this tutorial, the command that you should type will be preceded by the word “Command” and a colon.

Command: This is a Command Line

The example given in this tutorial will demonstrate the use of the internal geometry generation capability within CUBIT to generate a mesh on a perforated block. The geometry for this case is a block with a cylindrical hole in the center. The Create, Brick, Cylinder and Subtract commands are used to create solid model geometry with primitives and boolean operations. The block is then meshed using paving and translation. The geometry to be generated is shown in Figure 2-1. This figure also shows the curve and surface identification (ID) numbers of the

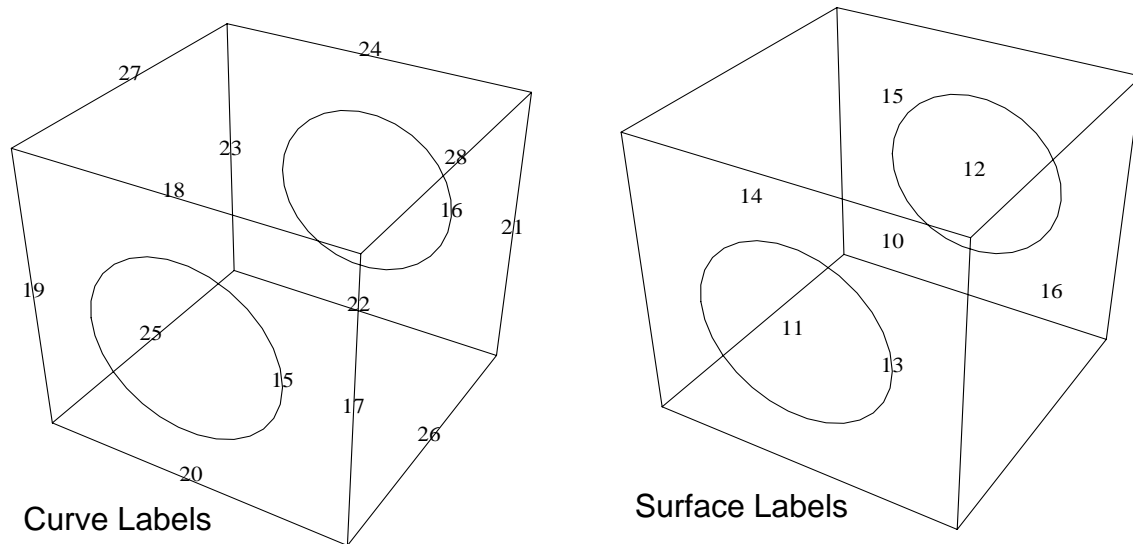


Figure 2-1 Geometry for Cube with Cylindrical Hole

geometry. These ID numbers are used in the command lines shown with each step. The final meshed body is shown in Figure 2-2 and also at the end of this chapter.

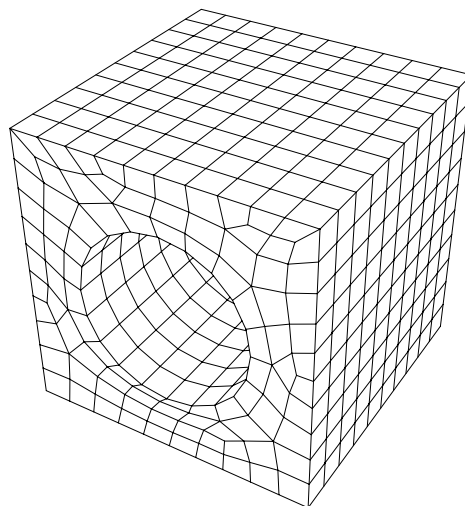


Figure 2-2 Generated Mesh for Cube with Cylindrical Hole

▼ Step 1: Beginning Execution

- Type “cubit” to begin execution of CUBIT. If you have not yet installed CUBIT, see instructions for doing so in the “CUBIT Installation” Appendix. A CUBIT console window will appear which tells the user which version is being run and the most recent revision date. (See the following screen shot for example of window). This window relays information about the success or failure of attempted actions.

```

      CCCCC      UU      UU      BBBBBB      IIII      TTTTTT
      CC      CC      UU      UU      BB      BB      II      TT
      CC      UU      UU      BB      BB      II      TT
      CC      UU      UU      BBBBBB      II      TT
      CC      UU      UU      BB      BB      II      TT
      CC      CC      UU      UU      BB      BB      II      TT
      CCCCC      UUUUU      BBBBBB      IIII      TT

[ *** CUBIT Version 1.8.1 ***

      *** ACIS Version 1.5 ***

      Revised 5/1/94

      AN ALL-QUADRILATERAL AND ALL-HEXAHEDRAL MESH
      GENERATION PROGRAM FOR
      PRE-PROCESSING OF FINITE ELEMENT ANALYSES

      CUBIT is based upon ACIS software by SPATIAL TECHNOLOGY INC.

      Executing on 05/20/94 at 09:37:35

```

- At the bottom of the CUBIT window you will be told where the commands entered in this CUBIT session will be journalled. For example: “Commands will be journalled to ‘cubit01.jou’.
- Below that, you will be offered the command line prompt: “CUBIT>”.
- Commands are entered at that prompt, followed by the “Enter” key.
- You should also see a blank graphics display window.

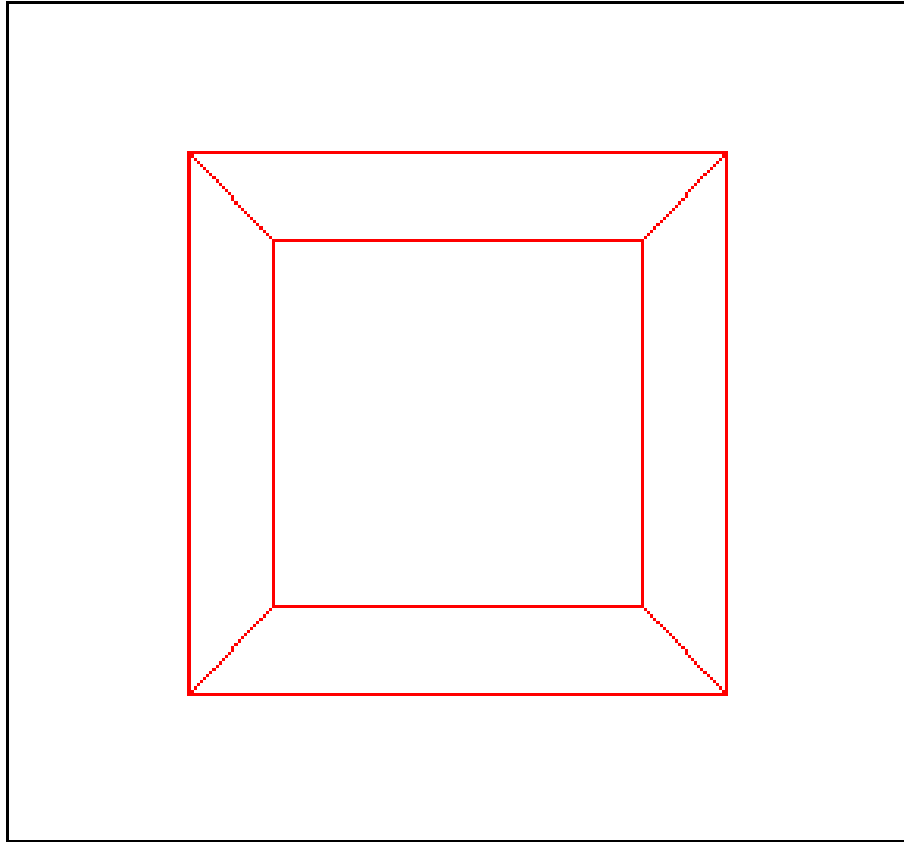
▼ Step 2: Creating the Brick

Now you may begin generating the geometry to be meshed. You will create a brick of width 10, depth 10 and height 10. The width and depth correspond to the x and y dimensions of the object being created. The “width” or x-dimension is screen-horizontal and the “depth” or y-dimension is screen-vertical. The height or z-dimension is out of the screen. The command to create an object is **Create**, followed by the type of geometry and its dimensions. Enter the following command.

Command: Create Brick Width 10. Depth 10. Height 10.

- The cube should appear in your display window as shown below.

Brick Display



- Note that the journalled version of the command is echoed above the next command line along with the confirmation message “brick body 1 successfully created.”
- The controller for the command line interface can tell from context when the user wants to create an object, so the command word **Create** can be left out. The same result as above would have been obtained by entering:

Command: Brick Width 10. Depth 10. Height 10.

- Try this after first issuing a **Reset** command as follows to remove all previous geometry and previous mesh from computer memory.

Command: Reset

- The command line interpreter can also recognize shortened versions of commands if they are unambiguous. The following command line would also have worked.

Command: Br Wi 10. Dep 10. Hei 10.

- If the command line parser is not able to find an unambiguous match to one of your command words, it will give an error message and offer a list of possible matches. For clarity, this tutorial will use unabbreviated command words.
- If the brick being created is a true cube, only the width needs to be specified, so the following command would also have worked. Try this after issuing a **Reset** command.

Command: brick width 10

- Notice that the command line is not case-sensitive, so **Brick** and **Width** do not need to be capitalized.

▼ Step 3: Creating the Cylinder

Now you must form the cylinder which will be used to cut the hole from the brick:

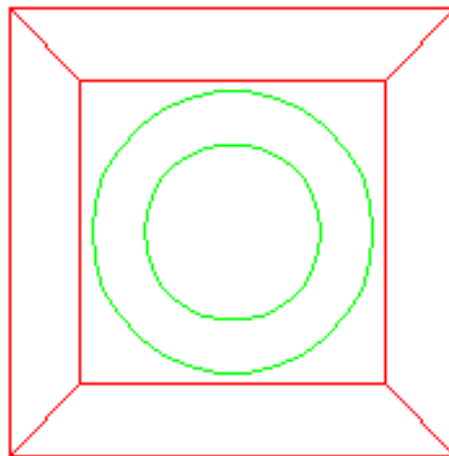
Command: create cylinder height 12 radius 3

or, by leaving the **Create** command as implicitly understood:

Command: cylinder height 12 radius 3

At this point you will see both a cube and a cylinder appear in the CUBIT display window.

Brick with Cylinder Display

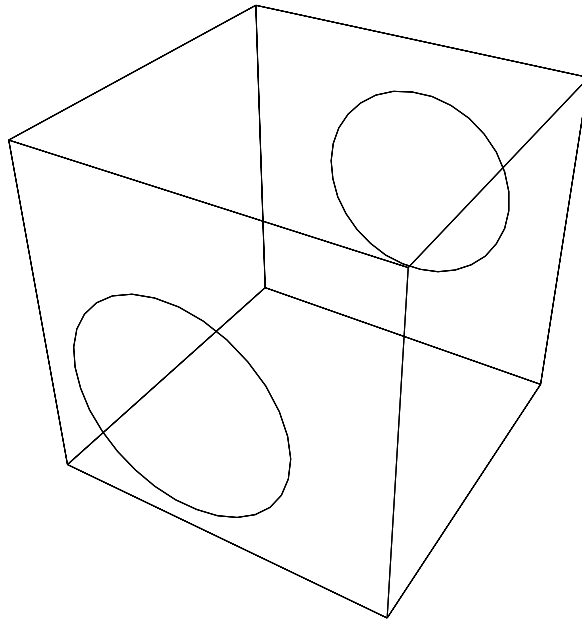


▼ Step 4: Adjusting the Graphics Display

The picture on the graphics display can now be adjusted to verify that what you expected to happen has indeed occurred. Issue the command

Command: from 3 4 5

This changes the viewpoint of the screen to a viewing location along the vector (3,4,5). The command word **From** stands for “view from.” The display should now look like the following figure.



At this point you may rotate the model to view its validity. The easiest method is to use the mouse. Type **Mouse** on the command line. You should see the message: “Entering mouse-based rotation/zooming/panning mode” and a circle should appear around the figure in the graphics window. To rotate the figure about its y-axis, position the mouse pointer outside the circle, hold the left mouse button down and move the mouse pointer around the circle. To rotate it around the x- and z-axes, position the pointer inside the circle and hold the left button down while moving the mouse. To exit the mouse-based mode of rotation, type **q**.

In the display, the wireframe picture shows the relative locations of the bodies. Turning the image to smooth shaded (as will be described in following steps) improves the perspective.

▼ Step 5: Forming the Hole

Now the cylinder can be subtracted from the brick to form the hole in the block. Issue the following commands.

Command: Subtract 2 From 1

Note: Note that both original bodies are deleted in the boolean operation and replaced with a new body (3) which is the result of the boolean operation **Subtract**.

▼ Step 6: Setting Body Interval Size

The next step is to generate the surface mesh on one of the surfaces to be swept. The number of increments must be set before meshing any geometry. This is done first for the entire body by specifying a desired size interval. Recall that body number 3 is a 10 by 10 by 10 cube with a cylindrical hole through it. We decide to specify an interval size of 1, which suggests 10 intervals on each side. Issue the following command.

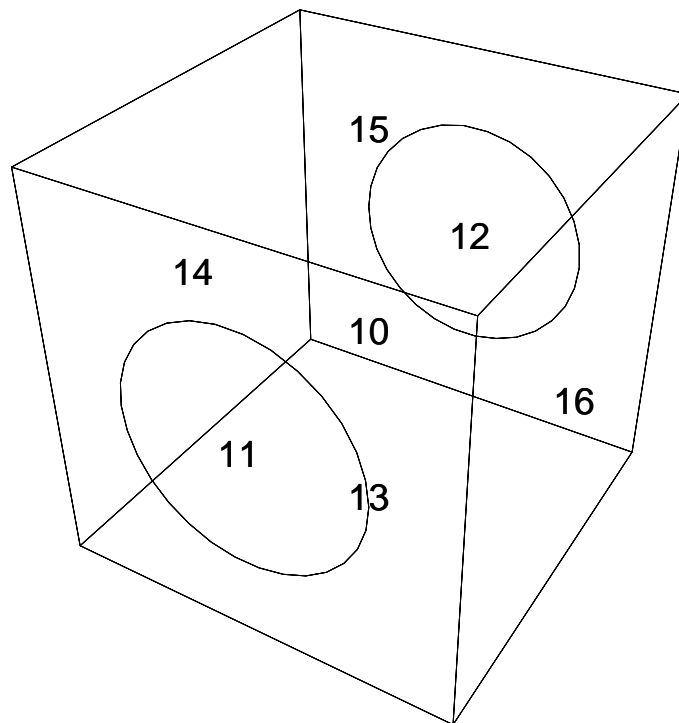
Command: body 3 interval size 1.0

▼ Step 7: Setting Specific Surface Intervals

The cylindrical surface (the inside of the hole) must be mapped in order for the sweeping type tools to work. Since this surface is periodic (contains no edge along the side of the cylinder) the mapping algorithm uses the surface interval setting to determine how many elements are to be mapped along the axis of the cylinder. The surface must first be identified. To see the surface numbers, issue the following commands.

Command: label surface on

Command: display



- The first command turns the surface labels on, but they do not become visible until the **display** command forces an update of the graphics screen. The surface labels can now be seen.

- The surface labels are positioned in the center of the geometric bounding box for each surface. From the display it is evident that the cylindrical surface is surface 10. To set the number of intervals to 10 on this surface only, issue the following command.

Command: surface 10 interval 10

▼ Step 8: Setting Specific Curve Intervals

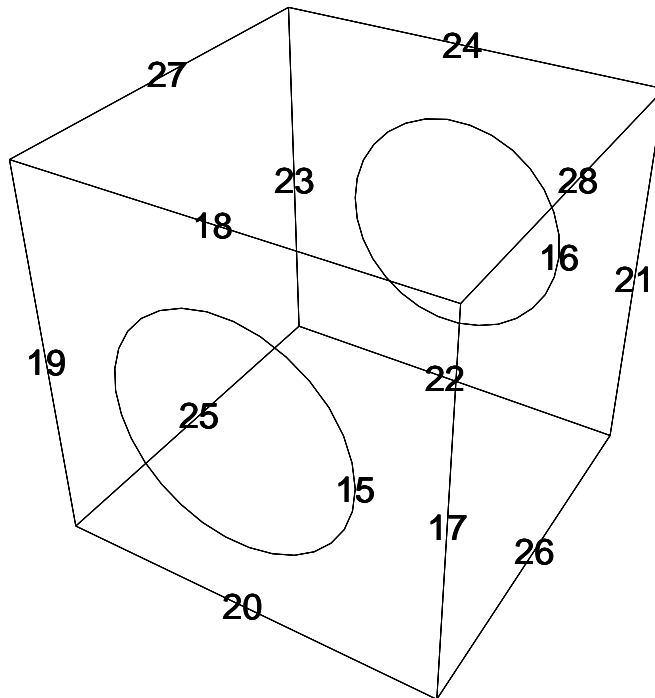
The surface interval command also propagates downward to the edges. However, in this case we want the circumference of the cylindrical ends of the surface to have 20 intervals instead of 10. This is accomplished by issuing a command of the form **Curve m interval n**, where m is the curve ID and n is 20. Before we can do that, we need to identify the curves. To turn off the surface labels and turn on the curve labels, issue the following commands.

Command: label surface off

Command: label curve on

Command: display

From the graphics display shown below, it is evident that curves 15 and 16 are the correct curve ID's.



To set the number of intervals on these curves to 20, issue the following command.

Command: curve 15 to 16 interval 20

Notice that we specified two curves, 15 and 16, by using the command syntax **15 to 16**.

▼ Step 9: Surface Meshing

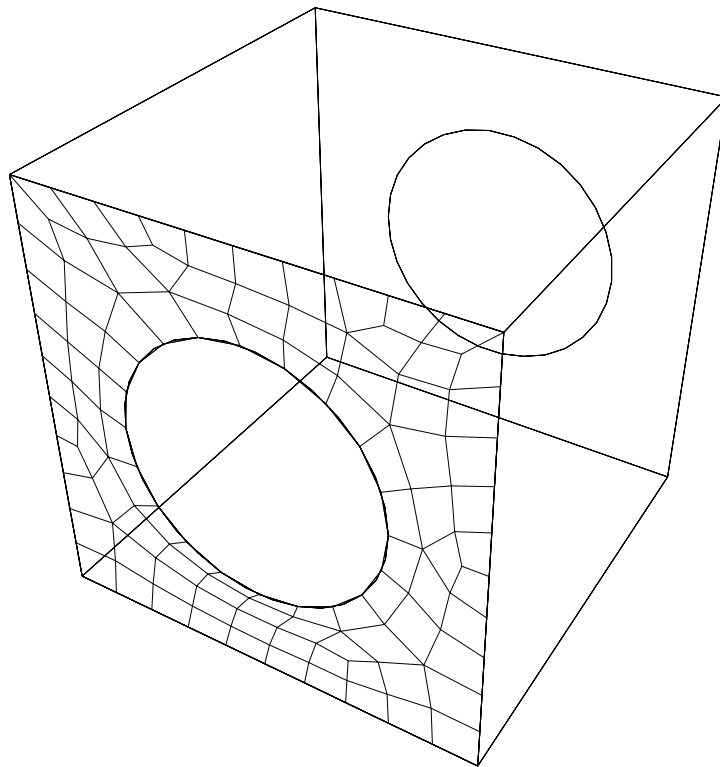
Now all necessary intervals have been set, and the meshing can proceed. Begin by meshing the front surface (with the hole) using the paving algorithm. This is done in two steps. First set the scheme for that surface to **Pave**, then issue the command to **Mesh**. Since the surface to be paved is number 11, issue the command:

Command: surface 11 scheme pave

With the meshing scheme specified, we proceed to mesh the surface.

Command: mesh surface 11

- The result of this meshing operation is shown below.



▼ Step 10: Volume Meshing

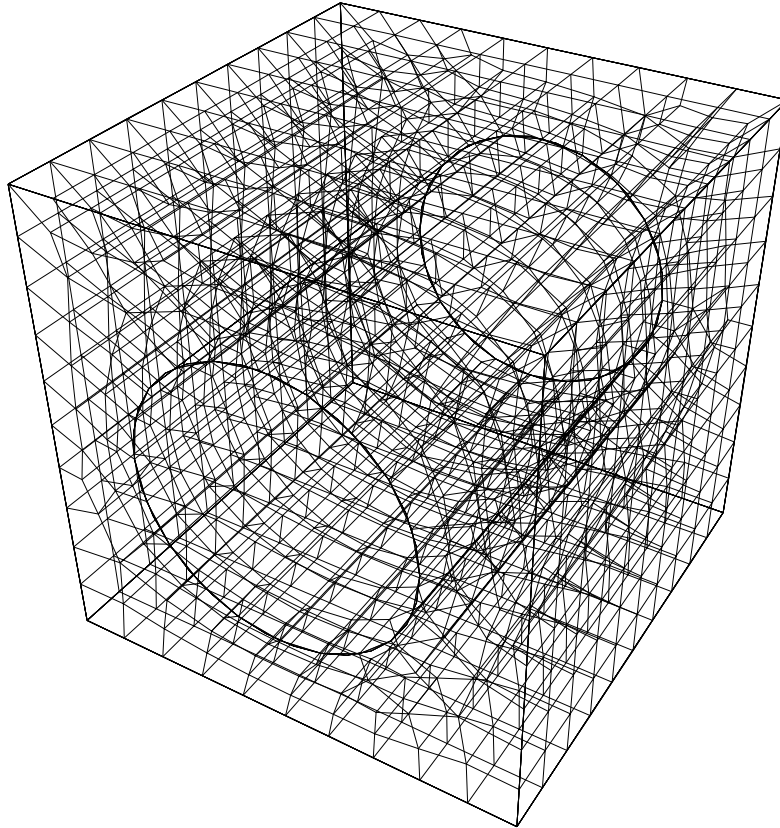
The volume mesh can now be generated. Again, the first step is to specify the type of meshing scheme and the second step is to issue the order to mesh. The scheme chosen is **translate**, which requires that source and target surfaces be specified. Issue the following command.

Command: volume 3 scheme translate source 11 target 12

With the scheme set, the **mesh** command may be given:

Command: mesh volume 3

The final meshed body will appear in the display window



The type, quality, and speed of the rendered image can be controlled in CUBIT by using several **graphics mode** commands, such as **Wireframe**, **Hiddenline**, and **Smoothshade**. For example:

Command: graphics mode hiddenline

Command: display

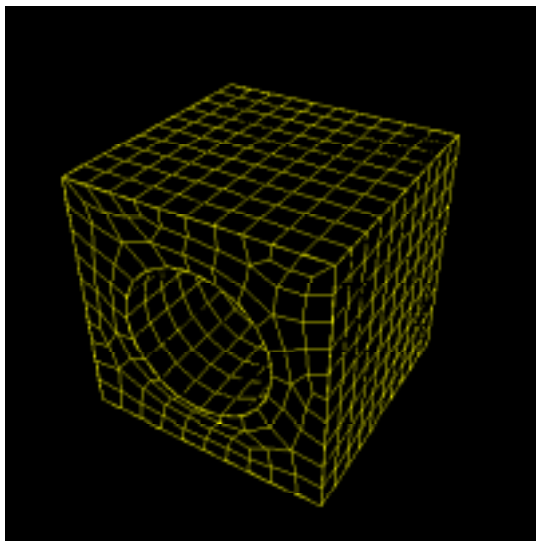
The hidden line display is illustrated below. Next, try:

Command: graphics mode smoothshade

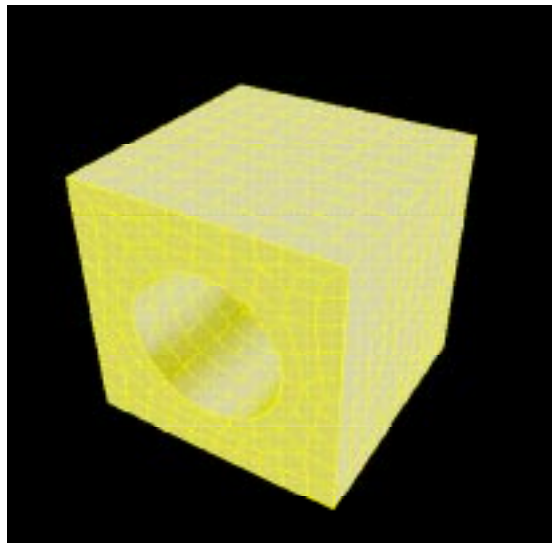
Command: display

The smoothshade display is also shown below.

For detailed information on these, see Chapter 3, Environment, “Image Rendering Control” and “Viewing the Model.”



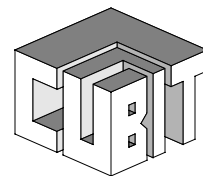
Hiddenline Display



Smoothshade Display

▼ Congratulations!

You have created your first CUBIT mesh. The following chapters contain more detailed information about using CUBIT and an in-depth description of the meshing algorithms available.



Chapter 3: Environment

- ▼ Interface Choices...39
 - ▼ Session Control...41
- ▼ Command Journalling...41
 - ▼ Graphics...42
- ▼ Model Information...52
 - ▼ Picking...60
- ▼ Help Facility...61

The CUBIT user interface is designed to fulfill multiple meshing needs throughout the analysis process. The user interface options include a traditional command line interface and batch mode operation. This chapter covers the interface options as well as the use of journal files, control of the graphics, a description of methods for obtaining model information, and an overview of the help facility.

▼ Interface Choices

Command Entry

CUBIT can be run in an interactive or batch mode. In interactive mode, commands are entered in the window from which CUBIT was executed (for information on the commands and options used to execute CUBIT, see “Execution Command Syntax” on page 19). In batch mode commands are read from a “journal” file, that is a file containing CUBIT commands. Commands can also be read from a file during interactive execution.

The command line interface provides the user access to all CUBIT commands via keyboard entry. When the command line version is executed, the command prompt **CUBIT>** appears in the UNIX shell window or terminal and a graphics window is created.

The CUBIT command line interface provides an EMACS-style line editing input package with command history¹. It allows the user to edit the current line and move through a list of previous

commands. Commands replayed from a journal file are not saved in the history list. The available editing commands are defined in Table 3-1.

Table 3-1 Command Line Interface Line Editing Keys

Key ^a	Function
^A, ^E	Move to beginning or end of line, respectively
^F, ^B	Move forward or backward one position in the current line.
^D	Delete the character under the cursor. Sends end-of-file if no characters on the current line.
^H, DEL ^b	Delete the character to the left of the cursor.
^K	Delete from the current cursor position to the end of the line
^P, ^N	Move to the previous or next line in the history buffer.
^L	Redraw the current line.
^U	Delete the entire line.
NL, CR ^c	Places current input on the history list, appends a newline and returns that line to the CUBIT program for parsing.
?	Provides “instant” help. If no text has been entered prior to typing the ‘?’, a list of all valid keywords will be echoed. If some text has been entered, either a list of all valid keywords matching the entered text is echoed, or if the entered text only matches a single keyword, the syntax for that keyword will be echoed. If the ‘?’ is entered inside the single or double quotes of a filename, all files (with the default suffix) matching the entered text will be echoed. The default suffixes are .sat for solid models, .jou for journal files, .fsq for FASTQ files, .ps for hardcopy files, and .g for mesh files.

- a. The notation ^X refers to holding down the control key and then typing the letter X. Case is not significant.
- b. See the documentation for your keyboard/workstation to determine which key sends the DEL character.
- c. NL is a newline, typically ^J, CR is a carriage return entered the normal way you end a line of text.

No-Graphics Interface

The CUBIT code is available in a form, typically called **cubitb** or the “no-graphics” version, that performs no graphical display. Except for that characteristic, **cubitb** is identical to the graphics version. The no-graphics version of CUBIT is invoked by entering ‘**cubitb <journal_file>**’ at the UNIX prompt¹. The EMACS-style line editing input package described in the previous section is also available in the no-graphics version.

1. The command line interface package used in CUBIT is Copyright 1991 by Chris Thewalt. The following copyright notice appears in the source code: “Permission to use, copy, modify, and distribute this software for any purpose and without fee is hereby granted, provided that the above copyright notices appear in all copies and that both the copyright notice and this permission notice appear in supporting documentation. This software is provided “as is” without express or implied warranty”.
1. See “Executing CUBIT” on page 19 for more information.

Batch Execution

Both the graphics and no-graphics version of CUBIT can be run in batch mode. That is, these versions of the code can be run taking commands from a command file and terminating execution when the end of the command file is reached. To initiate unattended operation, use the **-batch** option with the **cubit** or **cubitb** command, and specify a journal file from which commands are read. The journal file should contain the **Export** command to enable CUBIT to write out the model to a Genesis database file.

▼ Session Control

Several commands are available to control the overall CUBIT environment:

- **Exit.** The CUBIT session can be discontinued with either of the following commands
Exit
Quit
- **Reset.** A reset of CUBIT will clear the CUBIT database of the current geometry and mesh model, allowing the user to begin a new session without exiting CUBIT. This is accomplished with the command
Reset
- **Version** To determine the software version number, execute the **version** command. This command reports the CUBIT version number, the date and time the executable was compiled, and the version numbers of the ACIS solid modeler and the HOOPS library linked into the executable. This information is useful when discussing available capabilities or software problems with CUBIT developers.
- **Command echo** The echo of commands is controlled with the **[set] echo {on | off}** command. By default, commands entered by the user will be echoed to the terminal. The command **[set] logging {on | off} file 'filename'** can be used to additionally log all information output by CUBIT, including the command echo, to the file specified by **filename**.

▼ Command Journalling

Command journalling is used as a mechanism for saving sequences of CUBIT commands and a means to control CUBIT from simple text files. Command or “journal” files can be created within CUBIT, or can be created and edited directly by the user. They can also serve as a means of user support by reproducing commands leading to a code error.

CUBIT Journal File Generation

By default, the journalling facility is on. A journal file is created automatically in the current directory; the name of the journal file begins with the word “cubit” followed by a number between 01 and 99, followed by the characters “.jou”. The first few journal file names are **cubit01.jou**, **cubit02.jou** and **cubit03.jou**. The number following “cubit” will increment as more journal files are generated in that directory.

If no journalling is desired, the user may issue the command **[set] Journal Off** or run CUBIT with the **-nojournal** command line option. Turning journalling off should be done with care, as the journal file can save model re-creation time when errors occur during a long session.

Most CUBIT commands entered during a session are saved. The exceptions are commands that require interactive input (**mouse**, **pick**, **zoom cursor**), and the **play** command.

In addition to the default journalling, specific command sequences can be saved to user assigned files. To begin recording CUBIT commands to a file, use the command

Record '<filename>'

Once initiated, all commands issued in CUBIT are copied to this file, as well as to the default journal file (if journalling is turned on). Command recording to the record file is terminated with the command

Record Stop

The record command is particularly useful when a new finite element model is being built and alternate meshing strategies are being tested. Once the geometry has been defined, the record option can be used to record initial meshing controls and subsequent meshing commands. The mesh can be deleted, the recording terminated, and the process repeated to test alternate meshing strategies. To compare trial results, the user need only delete the current mesh and replay the journal file of the trial being considered.

Replaying Journal Files

To replay a journal file, issue the command

Playback '<filename>'

The file will be read and commands in the file executed. **Pause** commands can be inserted in the journal file to cause the command execution to pause at that point. Typing a return after the **pause** command will continue execution. Playback commands can be nested. Note that the filename must be enclosed in single quotes.

▼ Graphics

The graphics display window displays a graphical representation of the geometry and/or the mesh. The quality and speed of rendering the graphics, the visibility, location and orientation of objects in the window, and the labeling of entities can all be controlled by the user.

The geometry and mesh can be viewed from various camera positions and drawn in various modes (wireframe, hiddenline, shaded, etc). This section will discuss: 1) the control of the graphics window(s), 2) the control of the rendering parameters which affect the type, quality and/or speed of rendering of the image, 3) the control of which objects to draw and the color of drawn objects, 4) the desired labeling of objects on the image, 5) obtaining hard copy (e.g. postscript files) of the image, and 6) video animation generation.

Graphics Window Control

The graphics window is where the meshing graphics will be displayed and is the default viewport. The following attributes of the window can be controlled:

- **WindowSize.** The graphics window may be resized with the mouse, or with the commands

Graphics WindowSize Maximum

Graphics WindowSize <x_dimension> <y_dimension>

where **Maximum** will make the graphics window as big as the screen and the **x_dimension** and **y_dimension** are given in screen coordinates (pixels).

- **Background Color.** The window background color defaults to black but can be changed at any time using the command

Color Background <color_name>

Color Background <color_number>

where **color_name** is one of the colors listed in Appendix D, and **color_number** is an integer ID identifying the color. The background color can also be set using the **Graphics** menu **Color** dialog box as explained in section “Color” on page 50.

Multiple graphics windows can be created if desired. If multiple graphics windows are generated, only one of the display windows is active at any point in time -- that is, user commands which affect the graphics display only affect the currently active window. The following user capabilities are provided:

- **Create Window.** Create a new window using the command

Graphics Window Create <window_id>

where **window_id** is an integer graphics window identifier in the range 1-9 (window 0 always exists). When a window is created, it is initialized to the default graphics state.

- **Delete Window.** Delete an existing window (note: window 0 cannot be deleted) using the command

Graphics Window Delete <window_id>

where **window_id** is the integer graphics window identifier of the window to be deleted.

Set Current Active Window. Select the graphics window which is to be currently active using the command

Graphics Window Active <window_id>

where **window_id** is the integer graphics window identifier of the selected graphics window (range 0-9).

Image Rendering Control

The type, quality, and speed of the rendered image can be controlled in CUBIT using several graphics mode types and rendering options. The graphics mode type is set by using the **Graphics Mode Type** option menu. The available graphics mode types are:

- **Wireframe.** Wireframe drawing is the quickest mode, but it also can be the most confusing if the mesh or the geometry is very complex. No hiddenline processing is done. The command to set this mode is

Graphics Mode Wireframe

- **HiddenLine.** This option produces an accurate hiddenline representation of the mesh or geometry. The command to set this mode is

Graphics Mode HiddenLine

- **PolygonFill.** This option is similar to flat shading, but uses a slightly different algorithm. The command to set this mode is

Graphics Mode PolygonFill

- **Painters**¹. This option produces a shaded image where each polygon is drawn in a single shade. The polygons are drawn in a depth-sorted order. Although a correct rendering is produced for most images, there are cases where an incorrect image may be rendered. This mode is usually faster than the FlatShade and SmoothShade modes. The command to set this mode is

Graphics Mode Painters

- **FlatShade.** This option produces images where each polygon is drawn in a single shade. The image is slightly degraded with this option, but the speed of rendering is improved. The command to set this mode is

Graphics Mode FlatShade

- **SmoothShade.** A smoothshaded image produces the highest quality picture, but at the most expense. Colors are blended continuously over the drawn surfaces. The command to set this mode is

Graphics Mode SmoothShade

- **Dual.** This mode is designed to show the dual of the mesh as generated during whisker weaving. With this setting, the outside element edges are drawn as wireframe and the whisker sheets are drawn in smooth shading. This allows for a nice image of the structure of the dual of a hexahedral mesh. The command to set this mode is

Graphics Mode Dual

Graphics mode options control details of how the image is controlled between displays, and the type of enhancements added to the regular drawing modes. All options default to **On** at the start of execution. The graphics mode options available in CUBIT are:

- **Autocenter.** This option automatically centers the model in the viewport. The command to set this option is

Graphics Autocenter {On | Off}

- **Autoclear.** This option automatically clears the graphics window between displays, or updates. The command to set this option is

Graphics Autoclear {On | Off}

- **Border.** This option draws a border around the current viewport. The command to set this option is

Graphics Border {On | Off}

- **Axis.** This option controls the display of the axis or coordinate triad. The command to set this option is

Graphics Axis {On | Off}

1. The terminology “painters” is used since it draws the scene similar to the method used by a painter who might paint closer objects over more distant objects.

- **LineWidth.** This option controls the width of the lines used in the wireframe and hiddenline displays. The command to set the line width is

Graphics LineWidth <width>

- **Text Size.** This option controls the size of text drawn in the graphics window. The size given in this command is the desired size relative to the default size. The command to set the text size is

Graphics Text [Size] <size>

All option settings take affect at the time they are selected, it is not necessary to apply the changes. The **Set View Parameters** is a short-cut method for popping-up the Graphics View Dialog Box described in the next section.

Two additional commands,

graphics clear

graphics center

are available from the command line to perform a one-time only clear of the graphics window or centering of the model in the viewport. They do not affect the setting of the autoclear and autocenter toggles.

Viewing the Model

Figure 3-1.shows a schematic of the variables that effect the view of the object. Adjusting these

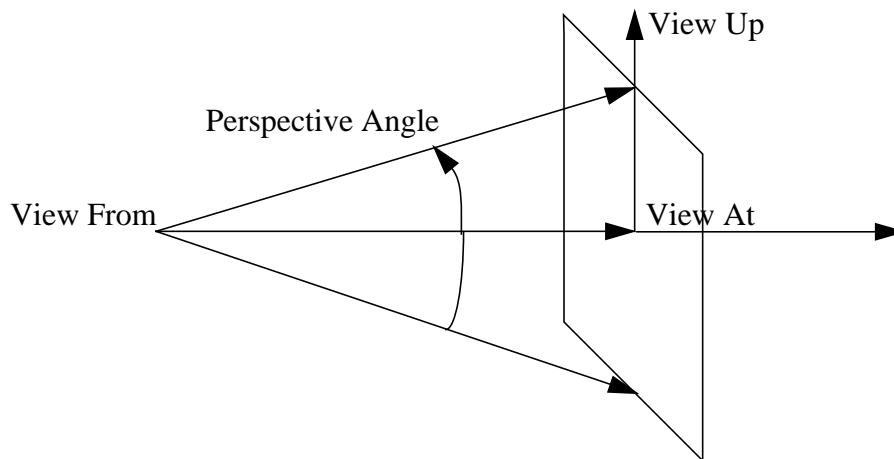


Figure 3-1 Schematic of From, At, Up, and Perspective Angle

variables will effect the way the three-dimensional model is projected onto the two-dimensional screen. These adjustments require you to update the display to see the results. The following adjustments can be made by the user:

- **View At Point.** The point you are viewing or looking 'at' can be set using the **X**, **Y**, and **Z** coordinates of the **View 'At'** text fields. To set the looking 'at' point using the command line, issue the command

[View] At <x> <y> <z>

- **View From Vector.** The point you are viewing ‘from’ can be set using the **X**, **Y**, and **Z** coordinates of the **View ‘From’** text fields. If automatic centering (see “Image Rendering Control” on page 43) is on, the input ‘from’ vector defines a relative viewpoint away from the ‘at’ point, in the direction of the ‘from’ vector. The magnitude of the ‘from’ vector is computed so that the picture fits nicely on the screen. When automatic centering is off, the ‘from’ vector defines an absolute viewpoint. To set the viewing ‘from’ vector using the command line, issue the command

[View] From <x> <y> <z>

- **Up Vector.** The up vector sets the orientation for the graphical display. In other words, along the line which connects the ‘from’ and the ‘at’ point, the up vector specifies which direction is displayed as up on the screen. This can be set using the command

[View] Up <x> <y> <z>

- **Rotate.** The rotation of the view can be specified by an angle about a world axis, or about a screen axis vector positioned at the focus point (Screen) or the camera. Additionally rotations can be specified about any general axis by specifying start and end points to define the general vector. The *right hand rule* is used in all rotations. To rotate about an axis, use the command

**[Rotate <angle> About [Screen | World | Camera] {X | Y | Z}
[Animation Steps <number>]**

The command defaults to the Screen coordinate system (rotations about a screen axis that is translated to the focus point). The Animation Steps option is included in this command (and all other rotation commands) to allow the user to perform a “smooth” rotation using several steps. This will let the picture appear to start and stop the total rotation *smoothly*. This is particularly useful when producing a video animation sequence (“Video Animations” on page 51) where a smooth sequence is desired. If the video system has been initialized, the animation command will take a snapshot at each step of the rotation.

Continuous rotations about any axis can be performed by double clicking one of the **+** or **-** buttons. The view is changed in sequential steps by the rotation increment specified. The screen is updated as fast as the picture can be processed. When the desired view is obtained the continuous rotations can be stopped by clicking the mouse somewhere in the View dialog box. These continuous rotations are not actually sent as commands to the parser, and as such are not stored in the journal file. To save the final state of the viewing angle, push the **Get Current Values** button in Figure 3-6 followed by the **Apply and Update** button. This will store the viewing parameters in the journal file. Continuous rotations are not available in the command line version.

Rotations can also be performed about the line joining the two vertices of a curve in the model, or a line connecting two vertices in the model. This is done with the commands¹

**Rotate <angle> About Curve <curve>
[Animation Steps <number>]**

**Rotate <angle> About Vertex <vertex_1> Vertex <vertex_2>
[Animation Steps <number>]**

- **Perspective.** The perspective angle can be set to adjust the relative perspective distortion of the view. A value of 0.0 will produce no distortion as if the viewing “from” location was at

¹1. See “Geometry Definition” on page 63 for definitions of Curve and Vertex

infinity. A larger value will produce more distortion. Values of about 15.0 degrees are normal. The perspective angle is set using the command

Graphics Perspective Angle <angle>

A more convenient method of adjusting the perspective is with a simple on/off toggle. This toggles the perspective angle between 0.0 and the current setting. The command to toggle perspective on and off is

Graphics Perspective {On | Off}

- **Zooming.** The image can be zoomed to provide a close-up view of portions of the image. The command used for mouse-based zooming is

[Graphics] Zoom Cursor

After entering the command, move the cursor to the graphics window and click the left mouse button at both corners of the desired zoom area. The command for performing a zoom is

[Graphics] Zoom <x_min> <y_min> <x_max> <y_max>

where the values specified are in screen coordinates (between 0 & 1). The **Zoom Reset** command updates the zoom limits to a size appropriate to capture all currently defined entities:

[Graphics] Zoom Reset

A simplified command line version of zoom has also been implemented that takes a single argument. The command syntax is defined as

[Graphics] Zoom Screen <scale_factor>

where `scale_factor` scales the view distance. Values of `scale_factor` > 1.0 zoom in toward the focus point while values of `scale_factor` < 1.0 zoom away out from the focus location. The best way to think of this is to think of a magnifying glass, a 2x zoom would magnify an object. Zooming can also be done on a particular entity in the model. For example a vertex or a curve can be zoomed in on using the command:

[Graphics] Zoom {group|body|volume|surface|curve|vertex} <entity_id>

The commands

List View

View List

will list the current values of the At point, From point, Up vector, and Perspective angle in the command line version. These are useful after doing multiple rotations, to journal a viewing angle

- **Mouse-based view manipulation.** The viewing parameters can also be modified interactively using the mouse. This is done by clicking and dragging the mouse while in “mouse” mode. While in mouse mode, the left mouse button is for rotating, the middle for zooming, and the right for panning. The best way to get a feel for how it works is to enter mouse mode and start experimenting. To enter “mouse” mode, enter the following at the CUBIT command line:

Mouse

Upon entering “mouse” mode, a list of options are displayed in the command line window. These options provide alternative ways of viewing the model while modifying the viewing

parameters. For example, if the model or mesh being viewed is very complex and detailed, the time required to re-display the model may take too long for the mouse-based manipulation to be effective. In this case, 'b' can be entered (while in mouse mode and while the mouse is anywhere in the graphics window) at the keyboard to toggle bounding box display on. When displaying in bounding box mode, only the bounding box or the model will be displayed while dragging the mouse in the window.

A similar command for 2D viewing is also available. This command only provides panning and zooming. The command is:

Mouse2D

Displaying Entities

The entities to be drawn in the current image can be controlled to limit the amount of information presented on the screen. There are two distinct modes of getting entities to the screen. The first is to set the visibility of the entities desired to be on and to turn the rest off. This visibility control establishes a "display list" of items that will be included in the image every time it is redrawn with a **display** command. The second method is an *immediate mode* drawing capability. Using a number of **draw** commands, individual items can be drawn onto the current picture. A draw command does not put the designated entities on the "display list" - it simply draws their wire frame image over the top of the current image. This immediate mode drawing is useful in highlighting specific nodes, faces, etc., but will not change the picture that is displayed when the image is updated using the **display** command. This section uses geometry and mesh terms defined in "Geometry Definition" on page 63, "Mesh Definition" on page 81, and "Finite Element Model Definition" on page 133. The reader may want to read those sections prior to reading the following discussion.

Drawing Entities

The series of **Draw** commands allow inspection of individual geometric and mesh entities. Individual entities or ranges of entities can be displayed. The draw commands affect the graphics system only temporarily and updating the display will show only those items actually in the display list.

The command line equivalent to draw selective entities is:

**draw {body | curve | edge | face | hex | volume |
node | nodeset | sideset | surface | vertex | group} <id_range>**

If **autoclear** mode is enabled, each **draw** command will clear the screen prior to updating the display. If **autoclear** mode is disabled, the specified entities will be added to the current set of displayed entities. An explicit **clear** command may be issued at any time to clear the display.

Highlighting Entities

An entity can be highlighted without erasing the remainder of the displayed model using the Highlight commands. These commands highlight the entity in the highlight color. The highlight commands available on the command line are:

highlight {body | curve | surface | vertex | volume} <id_range>

Currently, the highlight color is defaulted to a light gray.

Setting Visibility

Visibility of the geometry and the mesh is controlled by the use of global settings as well as through the use of individual (selective) geometric entity settings. For an entity to be visible, both the global setting and its individual visibility (if any) must be set on. In addition for mesh entities, the global mesh visibility flag, and the mesh visibility for the owning geometric entity, must be on as well.

• Global Settings

The following global settings can be used to adjust the visibility of groups of entities.

- **Geometry.** This controls the global visibility of geometry.
- **Mesh.** This controls the global visibility of mesh. If off, it overrides the specific global settings below.
- **Node.** This controls the visibility of mesh nodes (small dots). Defaults to off.
- **Edge.** This controls the visibility of mesh edges (line segments).
- **Face.** This controls the visibility of mesh faces (outlined by edges).
- **Hex.** This controls the visibility of mesh hexes (outlined by edges). Defaults to off.
- **Vertex.** This controls the visibility of geometric vertices (small dots). Defaults to off.
- **Hardpoint.** This controls the visibility of mesh nodes
- **NodeSet.** This controls the visibility of NodeSets (by nodes in the NodeSets). In order for NodeSets to be drawn, global node visibility must be on as well.
- **SideSet.** This controls the visibility of SideSets (by faces in the SideSets). In order for the SideSets to be drawn, global face visibility must be on as well.

The commands for setting global visibility flags are:

Geometry [Visibility] [on|off]

Hardpoint visibility [on|off]

Mesh [Visibility] [on|off]

{Hex|Face|Edge|Node} Visibility [on|off]

Node [Visibility] [on|off]

NodeSet [Visibility] [on|off]

SideSet [Visibility] [on|off]

Vertex [Visibility] [on|off]

• Individual Geometric Entity Settings

Two visibility flags are attached to individual geometric entities: 1) a flag indicating whether the geometry itself is to be included in the display list (visible), and 2) a flag to indicate if the mesh attached to the geometry is to be visible. For each geometric entity, the visibility of the item and any owned mesh can be set, or just the geometry visibility or the mesh visibility can be set. The visibility for bodies, volumes and surfaces can also be set with this interface.

The commands for setting selective visibility flags are:

{body | volume | surface | sheet | group} <range> visibility {on | off}

{body | volume | surface | group} <range> geometry visibility {on | off}

{body | volume | surface | group} <range> mesh visibility {on | off}

In addition, the visibility of mesh entities attached to genesis data can be controlled:

{Block | NodeSet | SideSet} <range> Visibility [on|off]

Color

The **Color** commands give the user customization control of the screen appearance of any geometric entity and its owned mesh entities. The default color used for an entity is the color of the owning entity¹. For example, if the color of a curve is not specifically set, it inherits the color of the owning surface. Mesh entity colors are determined by the owning geometry entity, unless set specifically according to the nodeset, sideset, or mesh entity color commands. The user can also control the color of the screen background.

The colors available at this time are listed in Appendix D. The commands to change colors are:

**color {body | volume | surface | nodeset | sideset | block | sheet | group}
<id_range> <color name>**

color {body | volume | surface | group} <id_range> mesh <color name>

color {body | volume | surface | group} <id_range> geometry <color name>

color {node | background} <color_name>

Entity Labeling

All geometric entities can be labelled with unique (to their geometric type) labels and, optionally the number of mesh intervals assigned to the entity, to enable specific entity identification. All mesh entities can also be labelled with unique (to their mesh entity type) labels to enable specific entity identification. In addition, the names of the geometric entities can also be used for the label. If the The labels are turned on or off by using the **Label** commands.

The labels will be displayed on the entity's centroid, which is helpful in the screen picking operations. The screen picks try to locate the entity with the closest centroid to the actual screen pick. Thus by turning entity labelling on, the user knows exactly where to click the pointing device in order to pick a specific entity. Labels are also useful in determining which entities were merged during a Feature Consolidation operation.

¹. See "Topology" on page 63 and "Mesh Definition" on page 81 for a description of the ownership of geometric and mesh entities.

The commands controlling labeling are:

label all {on | off}

label geometry {on | off}

label mesh {on | off}

label {body | volume | surface | curve | vertex} {on|off|name|interval|id}

Label {face | edge | hex | node} {on | off}

In addition, the size of text drawn to the graphics display, which includes entity labels, can be adjusted with the **Graphics Text Size** command; the command line syntax for this command is:

Graphics Text Size <size_factor>

where **<size_factor>** is a scaling factor relative to the default text size.

Hardcopy Output

The **hardcopy** command is used to capture graphics output to a PostScript or PICT file. Color or monochrome PostScript and encapsulated PostScript files are available. The commands for generating hardcopy output are:

hardcopy '<filename>' [encapsulated|postscript|eps] [color|monochrome]

Hardcopy '<filename>' pict [xsize <xpixels>] [ysize <ypixels>]

where xsize and ysize specify the pixel size of the created PICT image.

Video Animations



Several commands are available for generating a video recording of the graphics on the screen. The actual video initialization and recording has been set to work with the system in the graphics lab of the computational mechanics department. To initialize the video system, the commands

Video Initialize [Frames <frames>]

Video Initialize '<base_filename>' pict [xsize <xsize>] [ysize <ysize>]

are used. The first command will set up the recording devices, move the recorder to the first available frame, and set the system ready to record the picture displayed on the workstation screen next to the video recording equipment. The disk will be initialized to the specified number of frames or 2000 if not specified. It has been found that resizing the window with the following commands positions the graphics display in the proper position for optimal recording:

Graphics WindowSize Maximum

Graphics WindowSize 1170 820

The second command is used to create a separate PICT file of each frame. These PICT files can then be converted into a QuickTime or MPEG animation file using external software. The base_filename will have the frame number appended to the end to specify the sequence. The default PICT size is 640 pixels horizontal by 480 pixels vertical.

The video animation can now be generated by taking sequential snapshots of the screen. Each snapshot is captured by issuing the command:

Video Snap

Many of the meshing algorithms have been imbedded with flags that allow incremental snaps during the meshing process. These flags are activated by the initialization of the video device. This allows the generation of video animations of the meshing process rather easily. Often it is useful during an animation sequence to spin the picture around for the viewer to see the geometry and/or the mesh. These spins will appear choppy on the resulting video unless the object starts and stops rotation *smoothly*. The animated rotations described in the rotation section on page 46 is designed to provide this *smooth* rotation effect. These commands are also useful for showing the model to others during demonstrations. For video animations, 30 to 60 steps are needed when making a full revolution. If the video system has been initialized, the animation command will take a snapshot at each step of the rotation.

▼ Model Information

Information about the current CUBIT model can be obtained through the **List** commands. There are five general areas for which this information can be obtained: Model Summary, Geometry, Mesh, Special Entities, and Other. These are described in detail below. The descriptions will include sample output for some of the commands. To provide a frame of reference, the output will be for a 1x2x3 cube meshed with an average element size of 0.1 specified for the body. The journal file used to create the model is shown in Table 3-1.

Table 3-1 CUBIT Journal file used for List Output Examples

```
brick x 1 y 2 z 3
body 1 size 0.1
mesh volume 1
block 1 volume 1
nodeset 1 surface 1
sideset 1 surface 2
group "Surfaces" add surface 1 to 6
```

Model Summary Information

A brief summary of the current state of the model can be obtained from the list totals or list model command. The same information results from either command and provides information on the number of each type of geometric, mesh, and special entity in the current model. A sample output is shown in Table 3-2.

Geometry Information

The commands related to listing information about the geometry of the model are

list names [group|body|volume|surface|curve|vertex|all]

list {group | body | volume | surface | curve | vertex} [ids]

**list {group | body | volume | surface | curve | vertex} <id_range>
[geometry] [debug]**

The first command will list names currently used in the model and their corresponding entity type and id. If the **all** identifier is specified, all names in the model will be listed; if one of the other identifiers is entered, only names for that specific entity type will be output. Sample output

Table 3-2 Sample Output from ‘List Model’ Command

CUBIT> list model	
Model Entity totals:	
Geometric Entities:	
Total groups:	1
Total bodies:	1
Total volumes:	1
Total surfaces:	6
Total curves:	12
Total vertices:	8
Mesh Entities:	
Total Hex elements:	6000
Total mesh faces:	7876
Total mesh edges:	9854
Total mesh nodes:	7161
Special Entities:	
Total Element Blocks:	1
Total SideSets:	1
Total Nodesets:	1
Total BoundaryLayers:	0

from the **list names surface** command is shown in Table 3-3. This output shows that, for example, Surface 2 has the name ‘BackSurface’.

Table 3-3 Sample Output from ‘List Names’ Command

CUBIT> list name surf	
____Name_____	Entity Id
BackSurface	Surface 2
BottomSurface	Surface 3
FrontSurface	Surface 1
LeftSurface	Surface 4
RightSurface	Surface 6
TopSurface	Surface 5

The second command provides information on the number of entities in the model and their identification numbers. For large models in which several geometry decomposition operations have performed, this information is sometimes very useful. Sample output from the list surface ids command is show in the top portion of Table 3-4. For this simple problem, the information

is not very enlightening, output from a more complicated example is shown in the bottom portion of Table 3-4.

Table 3-4 Sample Output from ‘List Surface Ids’ Command

```
CUBIT> list surf id
There are 6 surfaces in the model. Their ids are:
    1 to 6

CUBIT> list surf id
There are 108 surfaces in the model. Their ids are:
    192 to 266, 268 to 271, 273 to 301
```

The second geometry-related information command provides more detailed information for each of the specific entities. This information will include the name and id of the entity, its meshed status, the number of owned mesh entities (if it is meshed), the settings for various meshing-related parameters (scheme, smooth scheme, size, and number of intervals), and a summary of the next lower-dimension geometry entities that make up this entity. The optional **geometry** identifier will additionally list the geometric bounding box for the entity. The optional **debug** identifier provides additional solid model information that is typically only of interest to developers of the geometry-related code. Table 3-5 through Table 3-10 show sample output for each of the group, body, volume, surface, curve, and vertex listing options.

Table 3-5 Sample Output from ‘List Group’ Command

Group Entity ‘Surfaces’ (Id = 1)					
Owned Entities:					
_____Name_____	_____Entity_____		Scheme/Meshed	Int	Int Size
FrontSurface	Surface	1	map/Y	1	0.100000
BackSurface	Surface	2	map/Y	1	0.100000
BottomSurface	Surface	3	map/Y	1	0.100000
LeftSurface	Surface	4	map/Y	1	0.100000
TopSurface	Surface	5	map/Y	1	0.100000
RightSurface	Surface	6	map/Y	1	0.100000

Table 3-6 Sample Output from ‘List Body’ Command

CUBIT> list body 1			
Body Entity ‘Body 1’ (Id = 1)			
Owned Volumes:			
_____Name_____	Id:	Meshed:	Use Count:
Volume 1	1	Yes	1

Mesh Information

The command related to listing information about the model mesh is:

list { hex | face | edge | node } <id_range>

The output from this command provides detailed information about the specified entities. This information will include the id of the entity, its owning geometry entity, and other entity-specific information. The hex output will indicate the Exodus Id¹, the volume which owns the hex element, and the eight corner nodes (in standard Exodus order). The face output lists the volume or surface which owns the mesh face, its four corner nodes, and a list of hexes that possibly share

Table 3-7 Sample Output from ‘List Volume’ Command

```
CUBIT> list volume 1
Volume Entity 'Volume 1' (Id = 1)

    Meshed:          Yes
    Mesh Scheme:     map (default)
    Smooth Scheme:    equipotential

Interval Count: 1
Interval Size:  0.100000
Block Id:       1
```

Owned Surfaces:		Mesh Scheme/			Interval:	
<u> Name </u>	Id	Meshed	Smooth Scheme	#	Size	
Surface1	1	map/Y	equipotential	1	0.100000	
Surface2	2	map/Y	equipotential	1	0.100000	
Surface3	3	map/Y	equipotential	1	0.100000	
Surface4	4	map/Y	equipotential	1	0.100000	
Surface5	5	map/Y	equipotential	1	0.100000	
Surface6	6	map/Y	equipotential	1	0.100000	

Table 3-8 Sample Output from ‘List Surface’ Command

```
CUBIT> list surf 1
Surface Entity 'FrontSurface' (Id = 1)

      Meshed:          Yes
      Total element faces:          200
      Total nodes (all inclusive):   231
      Mesh Scheme:      map (default)
      Smooth Scheme:    equipotential

      Interval Count: 1
      Interval Size:   0.100000
      Block Id:        0

      Total number of curves:  4
```

_____Name_____	Id	Scheme/ Meshed	Length	Interval: Number	Size	Factor	Vertices: Start,	End
Curve1	1	equal/Y		2	20 H	0.1	1/Y	2/Y
Curve2	2	equal/Y		1	10 H	0.1	2/Y	3/Y
Curve3	3	equal/Y		2	20 H	0.1	3/Y	4/Y
Curve4	4	equal/Y		1	10 H	0.1	4/Y	1/Y

this face. The edge output lists the volume, surface, or curve which owns the mesh edge, its two end nodes, the length of the edge, and a list of faces that possibly share this edge. The node output lists the coordinates and the volume, surface, curve, or vertex that owns the node.

1. The id of the hex when written to the Exodus database, not the CUBIT id. The default value is -1 before writing the Exodus database.

Table 3-9 Sample Output from ‘List Curve’ Command

CUBIT> list curve 1 to 12 by 2								
Name	Id	Scheme/		Interval:			Vertices:	
		Meshed	Length	Number	Size	Factor	Start,	End
Curve1	1	equal/Y		2	20 H	0.1	1/Y	2/Y
Curve3	3	equal/Y		2	20 H	0.1	3/Y	4/Y
Curve5	5	equal/Y		2	20 H	0.1	5/Y	6/Y
Curve7	7	equal/Y		2	20 H	0.1	7/Y	8/Y
Curve9	9	equal/Y		3	30 H	0.1	4/Y	7/Y
Curve11	11	equal/Y		3	30 H	0.1	3/Y	8/Y

Table 3-10 Sample Output from ‘List Vertex’ Command

CUBIT> list vertex 1 to 8						
Name	Id	Meshed	X-coord	Y-coord	Z-coord	
Vertex1	1	Yes	0.500000	-1.000000	1.500000	
Vertex2	2	Yes	0.500000	1.000000	1.500000	
Vertex3	3	Yes	-0.500000	1.000000	1.500000	
Vertex4	4	Yes	-0.500000	-1.000000	1.500000	
Vertex5	5	Yes	0.500000	1.000000	-1.500000	
Vertex6	6	Yes	0.500000	-1.000000	-1.500000	
Vertex7	7	Yes	-0.500000	-1.000000	-1.500000	
Vertex8	8	Yes	-0.500000	1.000000	-1.500000	

Table 3-11 through Table 3-14 show sample output for each of the hex, face, edge, and node options.

Table 3-11 Sample Output from ‘List Hex’ Command

CUBIT> list hex 1000 to 6000 by 1000							
Hex ID	Exodus ID	Owned By		Contains Nodes:			
1000	1000	Volume	1	2886	1358	28	126
				3057	1357	27	125
2000	2000	Volume	1	3741	1353	23	121
				3912	1352	22	120
3000	3000	Volume	1	4596	1348	18	116
				4767	1347	17	115
4000	4000	Volume	1	5451	1343	13	111
				5622	1342	12	110
5000	5000	Volume	1	6306	1338	8	106
				6477	1337	7	105
6000	6000	Volume	1	7161	1333	3	101
				691	690	1	82

Special Entity Information

Special entities include element blocks, sidesets and nodesets (boundary conditions), and boundary layers. Information specific to whisker weaving and dicing, including whisker sheets, whisker hexes, and dicer sheets, are also considered special entities. This information includes the number of mesh entities in the special entity and a list of the geometry entities owned by the special entity. Sample output for the list block, list sideset, and list nodeset commands are shown in Table 3-15, Table 3-16, and Table 3-17, respectively.

Table 3-12 Sample Output from ‘List Face’ Command

CUBIT> list face 1 to 7876 by 1000								
Mesh Face	Owned By		Nodal Connectivity			Shared by Hexes:		
1	Surface	6	1	3	101	82	6000	
1001	Surface	3	662	1072	1101	682	5820	
2001	Surface	5	2009	2010	2039	2038	3921	
3001	Volume	1	2143	2412	2583	2142	441	461
4001	Volume	1	1208	3646	3665	1237	1700	1900
5001	Volume	1	4577	1319	1318	4748	2960	2980
6001	Volume	1	5777	5776	602	573	4183	4383
7001	Volume	1	6638	6637	6808	6809	5389	

Table 3-13 Sample Output from ‘List Edge’ Command

CUBIT> list edge 1 to 9854 by 1000								
Edge	Owned By		Start/End Node		Length	Shared by Faces:		
1	Curve	10	1	3	0.100000	1	1271	
1001	Surface	6	543	542	0.100000	458	488	6331
2001	Surface	2	1047	1046	0.100000	954	974	2300
3001	Surface	4	1543	1542	0.100000	1458	1488	6230
4001	Surface	5	1992	2021	0.100000	1982	1983	3852
5001	Volume	1	2393	2222	0.100000	2830	2831	2836
6001	Volume	1	3551	3532	0.100000	3854	3855	4018
7001	Volume	1	4589	319	0.100000	4879	4880	4883
8001	Volume	1	5629	5458	0.100000	5905	5906	5909
9001	Volume	1	6668	6497	0.100000	6930	6931	6936

Table 3-14 Sample Output from ‘List Node’ Command

CUBIT> list node 1 to 7161 by 1000					
Node	X-coord	Y-coord	Z-coord	Owner	
1	0.500000	-1.000000	1.500000	Vertex	1
1001	-0.100000	0.400000	-1.500000	Surface	2
2001	0.200000	1.000000	1.300000	Surface	5
3001	0.200000	0.900000	-1.000000	Volume	1
4001	0.000000	-0.300000	-0.400000	Volume	1
5001	-0.100000	0.400000	0.200000	Volume	1
6001	-0.300000	-0.800000	0.800000	Volume	1
7001	-0.400000	-0.100000	1.400000	Volume	1

Table 3-15 Sample Output from ‘List Block’ Command

CUBIT> list block 1	
Block 1 contains 6000 3D element(s) of type HEX8.	
Owned Entities:	
Volume 1	Meshed

Other Information

Other non-geometry and non-mesh information is also provided through the list commands. These include message output settings, memory usage, and graphics settings.

Table 3-16Sample Output from ‘List SideSet’ Command

CUBIT> list sideset 1
SideSet 1: contains 200 element sides.
Owned Entities:
Surface 2 Meshed

Table 3-17Sample Output from ‘List NodeSet’ Command

CUBIT> list nodeset 1
NodeSet 1: contains 231 nodes.
Owned Entities:
Surface 1 Meshed

Message Output Settings

There are several types of messages output by CUBIT that are of interest to CUBIT users and developers. These messages and the type of information they convey are:

- **Information**- messages that contain information that is helpful to the user, but not critical to the operation of the program.
- **Warning** - messages that signal problems that may or may not be important to the operation of CUBIT.
- **Error** - messages signaling errors in the operation of CUBIT; these types of errors usually result in the termination of the program.
- **Debug** - debugging messages used by CUBIT developers.

The printing of Information, Warning and Debug messages can be turned on or off with the appropriate set command (Error messages are always printed). There are multiple Debug message flags, each controlling debug output for a different part of CUBIT. The value of each message flag or all message flags can be printed with the **List Settings** command. The commands used to print the value of message flags are:

list {echo | info | warning | journal | debug | settings}

Message flags can also be set using command line options; the Warning and Information flags are set with **-warning={on|off}** and **-information={on|off}** options, respectively. The Debug flags are set with **-debug=<setting>**, where **<setting>** is a comma-separated list of integers or ranges of integers. An integer range is specified by separating the beginning and the end of the range by a hyphen. For example, to set debug flags 1, 3, and 8 to 10 on, the syntax would be **-debug=1,3,8-10**. Flags not specified are off by default. Debug messages are typically of importance only to developers and are not normally used in normal execution.

The **List Settings** command lists the value of all the message flags, as well as the journal file and command echo settings; an example of the output from this command is shown in Table 3-18. The first several lines indicate the current settings of the debug flags, where the debug output will be output if the flag is on, and a short description of the purpose of the debug flag. For example, debug flag is enabled, its output will be written to the file ‘timing.log’ and the purpose of the flag is to output timing information.

Following the debug flag information is the settings of the echo, info, journal, and warning flags. Typically these should always be enabled. The final line of the output indicates whether logging is enabled and if so, where the information will be output. If logging is enabled, all echo, info,

warning, and error messages will be output both to the terminal and to the logging file. The other options to the list command select specific subsets of the list settings output.

The settings of the info, warning, journal, logging, and debug flags is set via the following commands:

```
[set] logging {on | off} file 'filename'
[set] {info | warning | journal} {on | off}
[set] debug <id> {on | off} terminal
[set] debug <id> {on | off} file 'filename'.
```

Table 3-18Sample Output from ‘List Settings’ Command

CUBIT> list settings			
Debug Flag Settings (flag number, setting, output to, description):			
1	OFF	terminal	User Interface: If flag is enabled, filenames being used for input will be echoed and each input line will be echoed prior to being parsed.
2	OFF	terminal	Whisker weaving information
3	ON	'timing.log'	Timing information for 3D Meshing routines.
4	OFF	terminal	Testing of video generation - if on then video specific drawing is enabled and related debug statements will be printed.
... (several lines deleted) ...			
45	OFF	terminal	Pillow Sheet debugging
46	OFF	terminal	Paver breakout detection (expensive)
echo	= On		
info	= On		
journal	= On		
warning	= On		
logging	= On, log file = 'test.log'		

Graphical Display Information

The list view command provides information about the current settings of various graphics parameters. Sample output from this command is shown in Table 3-19. See the description of the See “Graphics” on page 42. for a description of this information.

Memory Usage Information

Information about CUBIT’s memory usage can be obtained from the **list memory** command. An optional identifier can be specified which will restrict the output to the memory usage for those types of objects only. The command syntax is:

List Memory ['<object type>']

Sample output from the list memory command is shown in Table 3-20. This output is typically only of interest to CUBIT developers, so no interpretation of the output will be given here. If you need more details on this command, please contact one of the CUBIT developers..

Table 3-19 Sample Output from 'List View' Command

```
CUBIT> list view

...Current View Parameters
From: < 0.000000i 0.000000j 5.654066k> (absolute)
At: < 0.000000i 0.000000j 0.000000k>
Up: < 0.000000i 1.000000j 0.000000k>
View: < 0.000000i 0.000000j 1.000000k> (From - At, normalized)

Distance from 'from' to 'at' is 5.654066.
Displayed view size is 4.115823 horizontal by 4.115823 vertical.

Perspective Angle is 40.000000 degrees.
Drawing Mode is 'wireframe'.
AutoCenter ON, AutoClear ON, Axis OFF, Border ON.
```

Table 3-20 Sample Output from 'List Memory' Command

```
CUBIT> list memory

Dynamic Memory Allocation per Object
... (several lines deleted) ...
Object Name: DLLList

Object Size: 48      Allocation Increment: 4096

Allocated Objects: 4096 (bytes) 196608 (4% of Total)
Free Objects: 142 (bytes) 6816 (3%)
Used Objects: 3954 (bytes) 189792 (96%)

Object Name: ArrayMemory

Object Size: 32      Allocation Increment: 8192

Allocated Objects: 16384 (bytes) 524288 (12% of Total)
Free Objects: 2508 (bytes) 80256 (15%)
Used Objects: 13876 (bytes) 444032 (84%)

Total Memory Allocation Information (bytes)

Allocated Memory: 4153344
Free Memory: 358160 (8%)
Used Memory: 3795184 (91%)

Total non-pool ArrayBasedContainer memory allocation = 123338 (bytes)
Maximum non-pool ArrayBasedContainer memory allocated = 132415 (bytes)
```

▼ Picking

A limited command-line capability exists to pick geometric entities in the CUBIT model. The user issues the **Pick** command, then clicks on the entity to be identified; CUBIT then reports

the id number and name of the picked entity in the command window. The following picking commands are available:

Pick {curve | surface | lump | volume | body | dicersheet [list] [multiple]}

If the **[list]** option is used, information about that entity is listed in the command window as if the **List** command had been issued. If the **[multiple]** option is used, then multiple entities can be picked with one issue of the **Pick** command. When the **[multiple]** option is used, CUBIT is put into a mode where multiple picking can occur. To exit this mode, type 'q' at the CUBIT command line while the mouse is still in the graphics window.

▼ Help Facility

Two help systems are available in CUBIT: an window-based hypertext help facility and a text-based help facility. The user can access the hypertext help facility by typing **hyperhelp** and the text-based help facility by entering **help** or **?**. To attain further information about specific commands, the user can enter a specific keyword after **hyperhelp**, or specific keywords and identifiers before or after **help** or **?**.

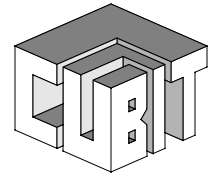
The text-based help facility prints the syntax for all commands that contain all of the specified keywords and identifiers to the standard output window. Unrecognized words and numbers are ignored. If no word is entered and recognized, the valid keywords are listed instead. For example:

```
CUBIT> volume 3 label fish ?
```

```
Help for words: volume & label
```

```
Label Volume [on|off|name|id|interval|size|merge|firmness]
Volume <volume_id_range> Label
               [on|off|name|interval|id|size|merge|firmness]
```

The window-based hypertext help facility is accessed by typing **hyperhelp <keyword>**. A window should pop-up containing a representation of the command index portion of this manual. Additional keyword parameters can be entered following the keyword to obtain more specific help. For example: **view at hyperhelp** would provide help on the **view at** command rather than general help on the **view** command. Your system must be configured to enable this capability; contact the CUBIT development team for configuration instructions.



Chapter 4: Geometry

- ▼ Geometry Definition...63
- ▼ Geometry Creation...66
- ▼ Geometry Manipulation...71
- ▼ Geometry Decomposition...75
- ▼ Geometry Consolidation...76

This chapter describes methods available in CUBIT to produce and manipulate the geometry needed for meshing. Definitions of geometric entities and the structure of the nonmanifold geometry represented by CUBIT are given. This is followed by sections describing geometry importation/exportation, creation and modification. Geometry consolidation, the process of generating cellular topology from a manifold model, and webcutting, the process of subdividing a three-dimensional solid into volumes that are each meshable, are also described.

▼ Geometry Definition

All geometric entities that exist in the CUBIT environment are represented by a solid model. In two-dimensional modeling systems, a list of connected directional line segments is sufficient to provide a complete and unambiguous geometric definition. In three dimensions, only a solid model representation can guarantee complete and unambiguous geometry. All meshing tools currently available in CUBIT use this solid model geometry when generating the discretized representation of the geometry, i.e., the mesh. ACIS[®] is the solid modeling engine currently used by CUBIT. However, CUBIT uses its own reference geometry (that overlays the ACIS[®] geometry) to represent a non-manifold cellular topology for mesh generation.

Topology

Topology refers to the manner in which geometric entities are connected within a solid model. Within CUBIT, the geometric entities consist of *vertices*, *curves*, *surfaces*, *volumes*, and *bodies*. A collection of one or more of these entities can be created and is called a *group*. They are defined as follows:

Vertex

A vertex occupies a single point in space. A vertex is used to bound a curve and to specify a location for a node. A vertex which is located in the interior of a surface is called a hard point. It is used to force a node to be located at that specific geometric position.

Curve

A curve is a line (not necessarily straight) which is bounded by at least one but not more than two vertices. An example of a single vertex curve might be the curve which bounds an end cap surface of a cylinder (both ends of the curve meet the same vertex). A curve is used to bound a surface. Curves may be generated independently of surfaces and can be used to specify the geometry for a sequence of (topologically one-dimensional) bar or beam elements.

Surface

A surface in CUBIT is a finite, bounded portion of some geometric surface (finite or infinite). A set of surfaces bound the material in a volume. Each surface is bounded by a set of curves. Surfaces must have a finite area and must at least be bounded by one curve to be meshed in CUBIT. Surfaces may be generated independently of three-dimensional (solid) volumes -- such free surfaces may be used to specify the geometry for shell elements. A periodic surface is a surface which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface has a discontinuity -- a jump from 0 to 2π -- in the periodic direction.

Volume

Volumes are volumetric regions and are always bounded by one or more surfaces. CUBIT currently cannot mesh a volume bounded by only one surface (e.g. a sphere) since such a surface has no bounding curves.

Body

A body is simply a set of volumes. It differs from volumes only in the fact that booleans are only performed between bodies, not between volumes. The simplest body contains one volume.

Group

A group is a collection of one or more topological entities (including other groups).

Groups

Groups provide a powerful capability for performing operations on multiple geometric entities with minimal input. They can also serve as a means for sorting geometric entities according to various criteria.

The command syntax to create or modify a group is:

```
group {id | "name"} add <list of topology entities>
```

For example, the command,

```
group "Exterior" add surface 1 to 2, curve 3 to 5
```

will create the group named **Exterior** consisting of the listed topological entities. Any of the commands that can be applied to the "regular" topological entities can also be applied to groups.

For example, **mesh Exterior**, **list Exterior**, or **draw Exterior**. A topological entity can be removed from a group using the command:

group <id> remove <list of topology entities>

When a group is meshed, CUBIT will automatically perform an interval matching on all mapped, submapped, and trimapped surfaces in the group (including surfaces that are a part of volumes or bodies in the group).

Grouping Sweepable Volumes

There is currently a limited capability to use groups to define the proper order of meshing for sweepable volumes. As is explained later in the section “Scheme Designation” on page 105, swept meshing relies on the constraint that the source and target meshes are topologically identical or the target surface is unmeshed. To help satisfy this constraint, a grouping method may be used to group all of the volumes that form a “sweeping chain”, or in other words, it groups the volumes in proper order of dependencies of meshing. For example, if the model was a series of connected cylinders, the proper way to mesh the model would be to sweep each volume starting at the top (or bottom) and continuing through each successive connected volume. With larger models and with models that contain volumes that require many source surfaces, the process of determining the correct sweeping ordering becomes tedious. To automate this the following command was added:

group sweep volumes

This command will search the model for volumes whose meshing schemes have been previously selected (either automatically or by the user), and group these volumes in a meshable order. The command will automatically generate groups of volumes which can be used to mesh the sweepable pieces.

Geometry Entity Identifier Ranges

The specification of geometric entities depends on identifying the objects by topology type, e.g. Vertex, Curve, etc. Geometric entity id ranges can be specified using the normal id range capabilities (see “Identifier Ranges” on page 23); id ranges for geometric entities

The following range parsing is also allowed for geometry entities:

<id_range> = all	Selects all identifiers for a given
-------------------------------	-------------------------------------

Cellular Topology

Cellular topology (a form of nonmanifold topology) allows the connection of any number of surfaces to a curve. A typical manifold or 2-manifold model allows a maximum of two surfaces to share a single curve. Cellular topology allows two adjacent volumes to share a common surface between them as shown in Figure 4-1. It also allows the formation of dangling faces and edges (Figure 4-2). These topological constructs are often required when generating meshes for complex geometries.

Cellular topology’s advantage for mesh generation is that, when used properly, it eliminates the problem of equivalencing the mesh entities that are supposed to be shared between adjacent geometric entities (for instance, the common surface, Web, shown in Figure 4-1). It also

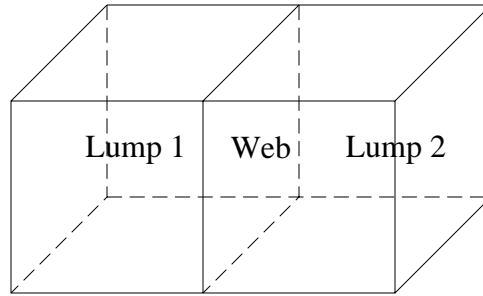


Figure 4-1 Cellular Topology Between Volumes

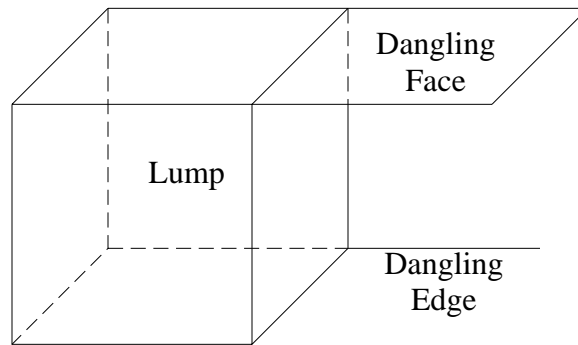


Figure 4-2 Dangling Faces & Edges

supports the formation of dangling faces and dangling edges as shown in Figure 4-2. This allows the proper connection of beam, surface, and solid mesh elements.

▼ Geometry Creation

The examples in “Examples” on page 159 show solid model construction using several standard techniques. One technique is to create primitives (e.g., bricks, cylinders, spheres, etc.) and to then perform booleans (e.g., subtractions, intersections, and unions) on the primitives to generate more complex geometric shapes. Another approach is to construct two-dimensional planar geometry and then perform sweep operations (either along a path or about an axis) to create three-dimensional solids. These geometry creation techniques can be combined to create arbitrarily complex geometries.

The creation of geometric primitives, their manipulation and positioning will be discussed first in this section, followed by a discussion of the boolean and sweep operations. Although CUBIT provides some geometry creation capabilities, it does not claim to be a geometric modeling package. In light of this, CUBIT allows the importation of geometry produced using several external programs. Several of these packages and the supported file formats are discussed later in this section.

Since the geometry acts as the template for discretization into a mesh, care must be taken to ensure an appropriate representation for the problem being analyzed. Users will find that building solid models of “real parts” will likely consist of a combination of the approaches mentioned above.

Geometric Primitives

The geometric primitives supported within CUBIT are pre-defined templates of three-dimensional geometric shapes. Users can create specific instances of these shapes by providing values to the parameters associated with the chosen primitive. For example, a cylinder is a basic shape or primitive. A user can create an instance of a cylinder by specifying the parameters associated with it -- *height* and *radius*. Primitives available in CUBIT include the brick, cylinder, torus, prism, frustum, pyramid, and sphere. These primitives can be generated and used in boolean operations to produce very complex shapes. The geometric primitives can also be used in boolean operations with geometry generated through other means (e.g., geometry read in from other sources). When using the GUI, the geometric primitives can be generated using the **Primitives** suboption in the **Geometry** menu. Figure 4-3 shows a sample of the available primitives.

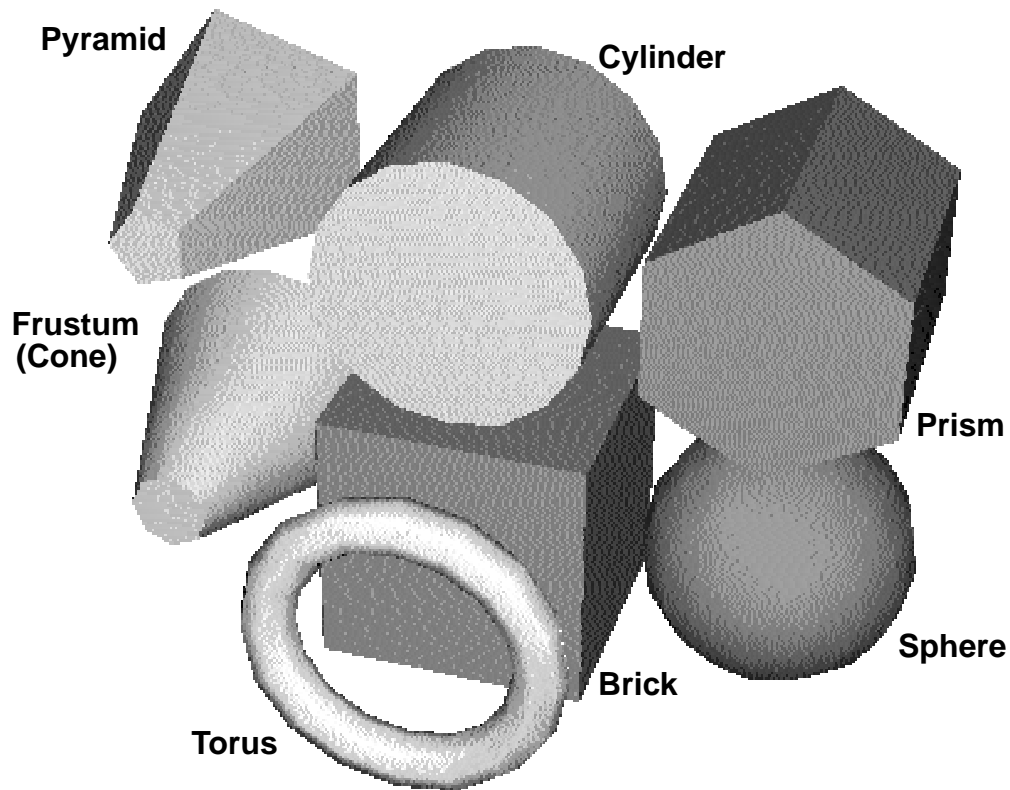


Figure 4-3 CUBIT Geometry Primitives

Brick

The brick is a rectangular parallelepiped. There are three parameters that may be specified, **Width** (x-dimension), **Depth** (y-dimension), and **Height** (z-dimension). Each of these must be strictly positive. If a parameter is left out, it defaults to the first of **Width**, **Depth**, or **Height** that was specified. In addition, a brick can be created with the dimensions a bounding box (approximately the smallest box containing a given set of entities). The commands to generate a brick primitive are:

```
[Create] Brick [{Width|X} <width>] [{Depth|Y} <depth>] [{Height|Z} <height>]
[ [Bounding Box] entity_type <id_range> ]
```

Any dimension explicitly given over-rides the dimensions of the bounding box, so for example it is possible to create a brick bounding a set of surfaces and edges lying in the xz-plane, but having a **Height** of 1 with the following command:

brick z 1 bounding box surface 1 4 5 edge 2

The new body contains one volume, They will be given id numbers one greater than the previously largest id. The brick will be aligned with the coordinate axes. The brick will be centered about either the origin or the center of the bounding box if one was specified.

Cylinder

The cylinder is a constant radius tube with right circular ends. There are two parameters that must be specified, **Height** (z-dimension), and **Radius** (x/y-dimension). The commands to generate a cylinder primitive are:

[create] cylinder height <z-height> radius <x/y-radius>

[create] cylinder z <z-height> radius <x/y-radius>

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin and aligned with the length of the cylinder along the z-axis.

Note: A cylinder may also be created using the **frustum** command with all radii set to the same value.



Prism

The prism is an n-sided, constant radius tube with n-sided planar faces around the tube. There are three parameters that must be specified, **Height** (z-dimension), **Sides** (number of sides) and **Radius** (x/y-dimension). The radius defines the circle circumscribing the prism cross-section. The commands to generate prism primitives are:

[create] prism height <z-height> sides <nsides> radius <x/y-radius>

[create] prism z <z-height> sides <nsides> radius <x/y-radius>

**[create] prism height <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius>**

**[create] prism z <z-height> sides <nsides> major [radius] <x-radius>
minor [radius] <y-radius>**

The new body contains one volume which will be given the next highest body ID number. The prism will be centered about the origin and aligned with the height of the cylinder along the z-axis. One of the planar, side (longitudinal) surfaces will be perpendicular to the X-axis. The number of sides must be greater than or equal to three.

Note: A prism may also be created using the **pyramid** command with all radii set to the same value.



Frustum

A frustum is a general elliptical right frustum. It can be thought of as a portion of a right elliptical cone. The elliptical nature comes by allowing a different radius in the two principle directions of the cone. There are four parameters that may be specified, **Height** (z-dimension),

Major Radius (x-radius), **Minor Radius** (y-radius) and **Top Radius** (x-radius at the top). The top y radius is calculated based on the ratio of the major and minor radii. If only **Height** and **Major Radius** are specified, the other two radii are defaulted to the major radius value. If all radii are equal, a frustum defaults to a simple cylinder. The commands to generate a frustum are:

```
[create] frustum height <z-height> major [radius] <x-radius>
                               [minor [radius] <y-radius> top <top-x-radius>]

[create] frustum z <z-height> major [radius] <x-radius>
                               [minor [radius] <y-radius> top <top-x-radius>]
```

The new body contains one volume which will be given the next highest body ID number. The frustum will be centered about the origin with the central frustum axis aligned with the z-axis.

Pyramid

A pyramid is a general n-sided prism. It can be thought of as a portion of a right elliptical cone. The elliptical nature comes by allowing different circumscribing radii in the two principle directions of the pyramid. There are five parameters that may be specified, **Height** (z-dimension), **Major Radius** (x-radius), **Minor Radius** (y-radius) and **Top Radius** (x-radius at the top). The top y radius is calculated based on the ratio of the major and minor radii given. If only **Height**, **Sides**, and **Major Radius** are specified, the other two radii default to the major radius value. If all radii are equal, a pyramid defaults to a simple n-sided prism. The commands to generate a pyramid are:

```
[create] pyramid height <z-height> sides <nsides> major [radius] <x-radius>
                               minor [radius] <y-radius> top <top-x-radius>

[create] pyramid z <z-height> sides <nsides> major [radius] <x-radius>
                               minor [radius] <y-radius> top <top-x-radius>
```

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin with the central pyramid axis aligned with the z-axis.

Sphere

The **sphere** command generates a simple sphere. Only one parameter may be specified, **Radius**. The command to generate a sphere is:

```
[create] sphere radius <radius>
```

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin.

Since portions of spheres are commonly required, the capability to generate hemispheres, quadrants, and octants is also provided. The command, which is an extension of the above command, is:

```
[create] sphere radius <radius> [inner [radius] <inner_radius>]
                               [delete] [xpositive] [ypositive] [zpositive]
```

If the inner radius is specified, a hollow sphere will be created with a void whose radius is the specified inner radius. The identifiers, **xpositive**, **ypositive**, and **zpositive**, specify which portion of the sphere will be retained, or if the **delete** identifier is present, the portion of the sphere that will be removed. For example, to create an hemisphere in the positive x direction, enter the command:

```
sphere radius 5 xpositive
```

Torus

The torus command generates a simple torus. Two parameters must be specified, **Major Radius**, or the radius of the spine of the desired torus, and **Minor Radius**, or the radius of the cross-section of the ring. The minor radius must be less than the major radius. The command to generate a torus is:

[create] torus major [radius] <major-radius> minor [radius] <minor-radius>

The new body contains one volume which will be given the next highest body ID number. It will be centered about the origin with the spline of the torus aligned perpendicular to the z-axis.

Importing Geometry

CUBIT can import geometry in several formats: ACIS[®] text (SAT) files, FASTQ input decks, and EXODUSII finite element data exchange format (with some limitations). Several commercial CAD software packages can generate ACIS[®] files directly, including MSC/ARIES, HP PE/SolidDesigner, and AutoDesk Inc.'s AutoCAD[®] Designer. Pro/Engineer models must be translated into the ACIS[®] SAT file format before being usable within CUBIT.

Importing ACIS Models

Externally-generated ACIS[®] files can be read into CUBIT using a single command. The command is:

Import Acis '<acis_sat_filename>'

Note that the filename must be enclosed in single quotes. This command will create as many bodies within CUBIT as there are bodies in the input file. Each CUBIT body will contain as many volumes as there are LUMPs in the corresponding ACIS body.

Importing FASTQ Models

Support is available for reading a FASTQ file directly into CUBIT. FASTQ files are imported into CUBIT using the **Import Fastq** command. The command is:

Import Fastq '<fastq_filename>'

Note that the filename must be enclosed in single quotes.

All FASTQ commands are fully supported except for the **Body** command (which is ignored, if present, as it is unnecessary), the "corn" (corner) line type, and some of the specialized mapping primitive **Scheme** commands. Standard mapping, paving, and triangle primitive scheme commands are handled. The pentagon, semicircle, and transition primitives are not handled directly, but are meshed using the paving scheme. The FASTQ input file may have to be modified if the **Scheme** commands use any non-alphabetic characters such as '+', '(', or ')'. Circular lines with non-constant radius are generated as a logarithmic decrement spiral in FASTQ; in CUBIT they will be generated as an elliptical curve.

Since a FASTQ file by definition will be defined in a plane, it must be projected or swept to generate three dimensional geometry. CUBIT supports sweeping options to convert imported FASTQ geometries into volumetric regions.

Importing EXODUSII Files

EXODUSII finite element data files can be imported under certain conditions. The capability to generate new geometry from deformed mesh is available for 2D EXODUSII files (4, 8, or 9

node QUAD element types) have do not have enclosed voids (holes surrounded by mesh) and have been originally generated with CUBIT and encoded with the **Nodeset Associativity** command. The **Nodeset Associativity** command records the topology of the geometry into special nodesets which allow CUBIT to reconstruct a new solid model from the mesh even after it has been deformed. The new solid model of the deformed geometry can be remeshed with standard techniques or meshed with a sizing function that can also be imported into CUBIT from the same EXODUSII file. CUBIT's implementation of the paving algorithm can generate a mesh following a sizing function to capture a gradient of any variable (element or nodal) present in the EXODUSII file.

All element blocks present in the EXODUSII file will be imported and represented with a different solid in the model. If the EXODUSII model contains interior voids or has undergone topology change during the analysis (“tearing” has occurred), an alternate method of constructing a CUBIT model is available through the **exofsq** external translator. This approach handles tearing (topology changes) and voids but requires a manual edit of the resulting FASTQ input deck to reapply the void. The **Import Free Mesh** command does not yet support models containing internal voids or tears.

The command line syntax to import EXODUSII data (can be deformed) is:

Import Free Mesh '<exodusII_filename>' Block <block_id> Time <time>

Note that the filename must be enclosed in single quotes.

Importing PRO/Engineer Models

The PRO/Engineer® product can also be used to create CUBIT geometry. Converters have been implemented which manage the translation of PRO/Engineer® assembly data into ACIS® format.

This solution is being pursued to address user needs and requirements regarding standard geometry formats at Sandia National Laboratories. Advantages to this creation method include the availability of the large number of parts being designed under the PRO/Engineer® format. PRO/Engineer® documentation and training is available at Sandia including consulting on an as-needed basis. Disadvantages include the single direction translation through which these parts must be sent to convert them to ACIS®. Some difficulties with numerical accuracy involving PRO/Engineer® have been cited although no negative impact of this on CUBIT meshing functionality has yet been observed.

Exporting Geometry

Geometry can also be exported from within CUBIT to other file formats. Currently, geometry can be exported to the ACIS SAT and DEBUG file formats. The SAT format can be used to exchange geometry between ACIS-compliant applications. The DEBUG format is merely a text file that describes the saved geometric models, and can be used for debugging purposes. The user can, optionally, specify which subset of bodies are to be exported. The commands are:

Export Acis '<acis_sat_filename>' [Body <body_id_range>]

Export Acis Debug '<acis_sat_filename>' [Body <body_id_range>]

Note that the filename is enclosed in single quotes. If the **Body** keyword is not specified, then all the bodies are saved.

▼ Geometry Manipulation

Bodies can be translated, rotated, reflected, scaled, and copied in order to position them correctly before performing other tasks associated with generating a model. Boolean, sweep and imprint operations can also be performed. The transform operations which translate, reflect, scale, rotate and copy are described first in this section, followed by a description of the boolean operations intersect, subtract and unite, and, finally, the sweep and imprint operations.

The translate, reflect, scale, and rotate functions do *not* create new geometry, whereas the copy, intersect, subtract, unite, sweep and imprint functions *do* create new geometry (except for the copy operation, the original bodies are removed and new bodies are created).

Transform Operations

Copy

The copy command copies an existing body to a new body without modifying the existing body. A copy can be made of several bodies at once, and the group can be translated with a specified offset, or rotated about a given vector. The commands for copying bodies are:

body <range> copy [move <x-offset> <y-offset> <z-offset>]

body <range> copy [reflect {x | y | z}]

body <range> copy [reflect <x-comp> <y-comp> <z-comp>]

body <range> copy [rotate <angle> about {x | y | z}]

body <range> copy [rotate <angle> about <x-comp> <y-comp> <z-comp>]

body <range> copy [scale <scale-factor>]

If the **copy** command is used to generate new bodies, a copy of the original mesh generated in the original body can also be copied directly into the new body. This is currently limited to copies that do not interact with adjacent geometry. For details on mesh copies, see “Mesh Importing and Duplicating” on page 127.

Move

The **move** command moves a body by a specified offset. The command to move bodies is:

Body <body_id_range> [Copy] Move <dx> <dy> <dz>

Body <body_id_range> [Copy] Move {x|y|z} <distance>...

If the **copy** option is specified, a copy is made and the copy is moved by the specified offset.

It is also possible to move bodies to absolute locations:

Move entity <id_range> location entity <id> [except {x} {y} {z}]

Move entity <id_range> location [x <val>] [y <val>] [z <val>] [except {x} {y} {z}]

Here entity is {vertex|curve|surface|volume|body}, and actually any combination of entities may be specified. This command moves the center of the entities to the specified location. (Note that bodies are integral, so moving an entity also moves all other entities that are in the same body.) “Except” is used to preserve the x, y, or z plane in which the center of the entity lies.

Scale

The **scale** command resizes the body without adding any new geometry. The body will be scaled about its centroid. The command to scale bodies is:

body <range> [copy] scale <scale>

If the **copy** option is specified, a copy is made and scaled the specified amount.

Rotate

The **rotate** command rotates a body about a given axis without adding any new geometry. If the **Angle** or any **Components** are not filled in they are defaulted to be zero. When using the command interface, a range of bodies to be rotated can be specified, as well as a rotation about one of the cartesian axes:

body <range> [copy] rotate <angle> about {x | y | z}

body <range> [copy] rotate <angle> about <x-comp> <y-comp> <z-comp>

If the **copy** option is specified, a copy is made and rotated the specified amount.

Reflect

The **reflect** command mirrors a body about a plane normal to an arbitrary vector without adding any new geometry. The commands to reflect bodies can be specified as well as directly specifying a plane normal to one of the coordinate axes:

body <range> [copy] reflect <x-comp> <y-comp> <z-comp>

body <range> [copy] reflect {x | y | z}

If the **copy** option is specified, a copy is made and reflected the specified amount.

Restore



The **restore** command removes all previous geometry transformations from the specified body. The command to restore bodies is:

body <range> restore

Boolean Operations

Boolean operators allow boolean interactions (e.g., intersection, union, etc.) between bodies to produce a new body. The boolean operators available in CUBIT for modifying bodies are intersect, subtract and unite.

Intersect

The **intersect** command generates a new body composed of the space that is shared by the two bodies being intersected. Both of the original bodies will be deleted and the new body will be given the next highest body ID available. The command is:

Intersect <body1_id> with <body2_id>

Subtract

The subtract operator will subtract one body from another. The order of subtraction is significant -- body2 is subtracted from body1. Both of the original bodies will be deleted and the new body will be given the next highest body ID available. The command is:

Subtract <body1_id> from <body2_id>.

Unite

The unite operator will combine two or more bodies into a single body. The original bodies will be deleted and the new body will be given the next highest body ID available. The commands are:

Unite <body1_id> with <body2_id>

Unite Body {<range> | all}

The second form of the command permits the uniting of multiple bodies in a single operation. If the **all** option is entered instead of a list of body ids, all bodies in the model will be united into a single body.

Sweep Operations

CUBIT now provides the ability to create and modify three-dimensional solids using the sweep operation. Sweeping of only planar surfaces, belonging either to two- or three-dimensional bodies, is allowed -- non-planar faces are not supported at this time.

In the following sweep commands:

The optional **draft_angle** parameter specifies the angle at which the lateral faces of the swept solid will be inclined to the sweep direction. It can also be described as the angle at which the profile expands or contracts as it is swept. The default value is 0.0.

The optional **draft_type** parameter is an ACIS-related parameter and specifies what should be done to the corners of the swept solid when a non-zero draft angle is specified. A value of 0 is the default value and implies an extended treatment of the corners. A value of 1 is also valid and implies a rounded (blended) treatment of the corners.

- This command allows the user to sweep a planar surface a specified distance in the direction specified by the vector:

```
sweep surface {<surface_id_range> | all}
                vector <x_vector y_vector z_vector>
                [distance <distance_value>]
                [draft_angle <degrees>]
                [draft_type <0 | 1>]
```

If the distance of the sweep is not explicitly provided, the face is swept a distance equal to the length of the specified vector.

- This command allows the user to sweep a planar surface about a specified axis that is in the plane of the surface being swept -- the angle of the sweep operation must be specified:

```
sweep surface {<surface_id_range> | all}
                axis {<xpoint ypoint zpoint xvector yvector zvector> | xaxis | yaxis | zaxis}
                angle <degrees>
                [steps <number_of_sweep_steps>]
```


[draft_angle <degrees>]

[draft_type <0 | 1>]

The **axis** of revolution must lie in the plane of the surfaces being swept. This axis can be chosen to be either the global coordinate axes (**xaxis**, **yaxis** or **zaxis**) or can be specified by a point (**xpoint**, **ypoint**, **zpoint**) and a vector (**xvector**, **yvector**, **zvector**). The **steps** parameter defaults to a value of 0 which creates a circular sweep path. If a positive, non-zero value (say, n) is specified, then the sweep path consists of a series of n linear segments, each subtending an angle of $[(\text{sweep_angle}) / (\text{steps} - 1)]$ at the axis of revolution.



Note: Specifying multiple surfaces that belong to the same body will not work as expected, as ACIS performs the sweep operation *in place*. Hence, if a range of surfaces is provided, they ought to each belong to a different body.

The sweep operations have been designed to always produce valid solids of positive volume, even though the underlying solid modeling kernel library that actually executes the operation, ACIS, allows the generation of solids of negative volume (i.e., voids) using a sweep.

Appendix B contains an example that demonstrates the use of the sweep operations to generate and modify three-dimensional solids.

Imprint Operation

It is important to maintain continuity of meshes across the bounding topological entities shared by neighboring bodies. This requires that when neighboring bodies abutt each other, the topology of the shared region in space must be identical on either side of this shared boundary. The process of merging these topological entities (described in the section, “Geometry Consolidation” on page 76) creates single entities that are shared by these neighboring bodies.

The imprint operation provides the user with the ability to ensure that identical topology exists where neighboring bodies abutt each other. The command is:

Imprint <body1_id> with <body2_id>

Such an **imprint** operation is often followed by a **merge** operation. This ensures that the duplicate topological entities created by the **imprint** operation between these two bodies are removed, thereby creating nonmanifold topological entities.

▼ Geometry Decomposition

The ability to decompose ACIS geometry now exists in CUBIT. This feature is required to facilitate the generation of a mesh for three-dimensional solids, as fully automatic mesh generation of arbitrary solids is not yet possible in CUBIT. The relevant commands are, **webcut** and **decompose**.

Web Cutting

Due to a decision not to use the ACIS Cellular Topology Husk, the decomposition of a body is effected through the use of boolean operations and results in the creation of new, manifold bodies. To facilitate mesh compatibility between neighbouring bodies, imprint operations are optionally performed to ensure that the topology between them matches -- this is the default.

Also, bodies are optionally merged at the end of a webcut operation, resulting in the creation of nonmanifold geometric reference entities, shared by neighboring bodies -- this is the default. It is important to note that the underlying ACIS bodies remain manifold throughout these operations.

New geometry is generated during a webcut operation and the original bodies are deleted. A body being webcut is decomposed into two new bodies -- one that is the result of a subtract boolean operation between the original body and the cutting tool body, and another that is the result of an intersect boolean operation between the original body and the cutting tool body.

In each of the **webcut** commands, the user specifies which set of bodies to be cut, a cutting tool (either as an existing body or by providing sufficient information to allow CUBIT to generate one), and a set of optional parameters specifying that the imprint and merge operations *not* be performed (by default, both these operations *are* performed after the boolean operations).

The syntax of the webcut commands are as follows -- each command provides the user with a different mechanism for specifying the *cutting tool*:

- The user can specify an *infinite cutting plane* by providing three non-colinear, existing vertices (to be able to perform the boolean operations, an infinite cutting tool body is created from this infinite cutting plane, as ACIS does not perform boolean operations on half-spaces):

```
webcut body {<body_id_range>|all}
      plane
      vertex <v1_id> vertex <v2_id> vertex <v3_id>
      [noimprint] [nomerge]
```

- The user can also provide an *infinite cutting plane* by specifying an existing, planar surface (to be able to perform the boolean operations, an infinite cutting tool body is created from this infinite cutting plane, as ACIS does not perform boolean operations on half-spaces):

```
webcut body {<body_id_range>|all}
      plane
      surface <surface_id>
      [noimprint] [nomerge]
```

- The user can specify that an existing body be used as the *cutting tool*. This cutting tool body is, itself, unaffected by the webcut operation, and is, unlike the other cutting tool bodies, not extended to be *infinite* in size:

```
webcut body {<body_id_range>|all}
      tool <tool_body_id>
      [noimprint] [nomerge]
```

Appendix B contains an example that demonstrates the use of webcutting operations.

Body-Based Decomposition

Primitive geometry decomposition can be performed using the geometry boolean operations available in CUBIT. An accelerated version of this type of operation is provided as well. If two ACIS bodies overlap in space, they can be decomposed into three separate bodies using the **Decompose** command (one body contains the overlapping region and the other two bodies contain the non-overlapping portions of the original bodies being decomposed):

```
Decompose <body_id> with <body_id>
```

The three bodies resulting from this operation will have manifold surfaces; geometry consolidation should be used if these surfaces are to be represented with a single mesh (see “Geometry Consolidation” on page 76).

▼ Geometry Consolidation

Geometry consolidation is a means of enforcing mesh continuity and avoiding mesh node equivalencing between neighboring bodies that abutt. When a manifold solid model is constructed, it may contain any number of volumes that can belong to a single body. This does not imply that these volumes are *aware* of each other -- CUBIT has no record or history of how the volumes from a body were created and what their spatial relationships may be. To determine these spatial relationships, some simple feature recognition concepts have been implemented to detect proximity between geometric entities, and to *consolidate* any redundant entities.

Such overlapping, redundant geometry, though not really redundant in a solid modeling sense, can cause problems when trying to generate a consistent mesh. From a meshing standpoint (in which a contiguous mesh is required between volumes) if two surfaces overlap spatially, the redundancy must be resolved by consolidating these two surfaces into a single one that is shared by the neighboring bodies. This requirement is illustrated in Figure 4-4. This figure depicts an “L” block, which can easily be meshed with a sweep operation. However, for the purposes of demonstrating the utility of geometry consolidation, we will first decompose it into two bodies using the **webcut** command (the result of the operation is shown in Figure 4-5).

For a contiguous mesh, CUBIT requires adjacent volumes to share identical surfaces where they meet. The geometry consolidation tool in CUBIT is designed to recognize the spatial equality of surfaces that overlap exactly and force them to use one surface mesh between them. This mesh sharing approach avoids the need to equivalence nodes and preserves the required mesh continuity. After the model in Figure 4-4 is decomposed into the model in Figure 4-5, surfaces 1 and 2 meet the *spatially equivalent* requirement. They can now be consolidated and treated as a single surface for meshing purposes. For the surfaces to be deemed *spatially equivalent*, the curves and vertices that constitute the boundaries of the consolidated surfaces need to be *spatially equivalent*, as well.

Note: Assemblies that are imported from external solid modelers (e.g., Aries[®] ConceptStation, PRO/Engineer, and the ACIS[®] Test harness) will need to be inspected to ensure that the adjacent solids have matching topology. Some of these external modelers use internal geometry engines other than ACIS[®] (e.g., PRO/Engineer does not utilize ACIS[®]); when using geometry that was created in PRO/Engineer, it is critical to set the accuracy within PRO/Engineer to the highest level possible before creating the geometry that is subsequently to a file. The standard level of accuracy within ACIS[®] is much higher than that of PRO/Engineer (10^{-6} vs. 10^{-3}). CUBIT relies on simple geometric reasoning to identify matching surfaces and curves, and this technique is sensitive to the accuracy of the models being processed. Since Aries[®] ConceptStation is based on ACIS[®], it does not suffer from accuracy problems due to varying geometric tolerances.

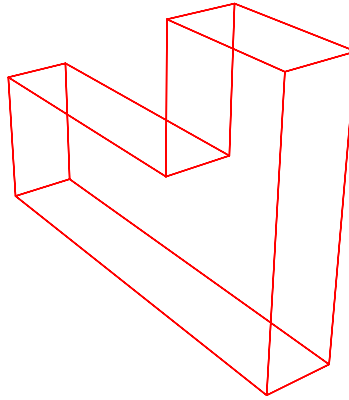


Figure 4-4 Solid Model Prior to Decomposition

General Geometry Consolidation

When a model fully complies with the matching topology requirement, it is ready for full geometry consolidation of redundant geometric entities. The **Merge all** command searches all bodies in the model for matching surfaces, then for matching curves, and finally for matching vertices. When a match is found, CUBIT *merges* the two matching entities in its model database.

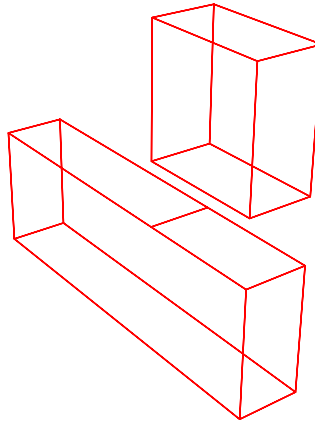


Figure 4-5 Solid Model After Decomposition

During such a merge operation, one of the topological entities being merged is removed from the database (typically the one with a higher numerical ID value) and the other is retained. All topological entities that previously referred to the one that was removed will be modified to now refer to the one that was retained.

Note: It is critical to complete geometry consolidation before any meshing of the affected geometry is initiated. If geometric entities are merged subsequent to meshing operations, some of the associated mesh entities may be lost.

General geometry consolidation is performed with the following commands:

Merge all

Merge body <body_id> with [body] <body_id>

Merge body <body_id_range>

The first command merges the topological entities of *all* bodies in the CUBIT model, whereas the latter two commands allow the user to merge the topological entities of a specified subset of the bodies in the CUBIT model.

Selective Geometry Consolidation

When models need to retain redundant geometric entities (such as redundant curves that are required to model slide lines), a more selective mode of geometry consolidation needs to be used. Selective geometry consolidation can be initiated between any set of user-specified surfaces or curves. The consolidation process will be limited to the specified entities and the entities connected to them.

The commands for selective geometry consolidation are as follows.

Merge curve <curve_id> with [curve] <curve_id>

Merge curve <curve_id_range>

Merge all curves

Merge surface <surface_id> with [surface] <surface_id>

Merge surface <surface_id_range>

Merge all surfaces

▼ Geometry Attributes

Each topological entity has attributes attached to it. These attributes specify aspects of the entity such as the color that entity is drawn in and the meshing scheme to be used when meshing that entity. This section describes those geometry attributes that are not described elsewhere in this manual.

Entity Names

Topological entities (including groups) are assigned integer identification numbers in CUBIT in ascending order, starting with 1. Each new entity created within CUBIT receives a unique id. However, topological entities can also be assigned names, and these names can be propagated explicitly by the user. The following command assigns names to topological entities in CUBIT:

{Group|Body|Volume|Lump|Surface|Curve|Vertex} Name '<entity_name>'

Each topological entity is given a unique default name when it is first created. Its default name consists of the name of the corresponding topological entities (body, volume, surface, curve, or vertex), followed by the ID number of the entity. For example, curve number 21 will have a default name, "curve21". The name of each topological entity appears in the output of the **List** command.

Topological entities can be identified either by the entity type followed by an identification number or by a unique name. Such a name can be used anywhere that an entity type and id may be used. For example, if surface 3 is named CHAMFER1, the command “mesh CHAMFER1” has the same result as the command “mesh surface 3”.

A topological entity may have multiple names, but a particular name may only refer to a single entity. If the geometry is imported from an ACIS SAT file that has the attributes **attrib-gtc-name**¹, the name of the corresponding CUBIT entity will be set to the text string stored in that ACIS attribute.

Note: In a merge operation, the names of the deleted entity will be appended to the names of the surviving entity.

The commands are:

Label {body|volume|surface|curve|vertex} {name|id|on|off}

Label geometry {name|id|on|off}

Label {name|id|on|off}

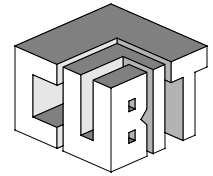
Control the type of labels displayed for an entity. If **name** is specified, the entity's name will be used in the display; if **id** or **on** are specified, the entity's id number will be displayed. The second and third forms of the command specify the labeling format for all geometry entities with a single command. The name of an entity can be set using the command

name {group|body|volume|surface|curve|vertex} <id> `entity_name`

A list of all names currently assigned and their corresponding entity type and id (optionally filtered by entity type) can be obtained with the command

list names [{group|body|volume|surface|curve|vertex|all}]

1. The attribute used to specify the names in the ACIS SAT file will probably change in the near future.



Chapter 5: Mesh Generation

▼ Mesh Definition...	81
▼ Mesh Attributes...	82
▼ Curve Meshing...	87
▼ Surface Meshing...	91
▼ Volume Meshing...	105
▼ Mesh Editing...	124
▼ Mesh Importing and Duplicating...	127
▼ Mesh Quality...	128

The methods used to generate a mesh using existing geometry are discussed in this chapter. The definitions used to describe the process are first presented, followed by descriptions of curve, surface, and volume meshing techniques. The chapter concludes with a description of the mesh editing and other miscellaneous capabilities.

▼ Mesh Definition

The mesh consists of entities similar in hierarchy to the geometry described in the previous chapter. The mesh entities include *nodes* (locations in space), *edges* (bar elements), *faces* (quadrilateral or shell elements), and *bricks* (hexahedral elements). Each mesh entity is associated with a geometry entity which owns it. This associativity allows the user to mesh, display, color, and attach attributes to the mesh through the geometry. For example, setting a mesh attribute on a surface affects all mesh entities owned by that surface.

Mesh Hierarchy

Mesh hierarchy refers to the manner in which mesh entities are connected within the mesh. Once a mesh is formed, the mesh entities define a discretized version of the geometry. The nodes required by higher-order elements are generated subsequent to the initial discretization; however, they are generated in the correct position based on the underlying geometry.

Node

A node is a single point in space. A node may be owned by (contained by) a vertex, curve, surface, or volume.

Edge

An edge is defined by a minimum of two nodes. Additional nodes may exist on the edges of higher-order elements. An edge may be owned by a curve, surface, or volume.

Face

A face is defined by four connected edges. A face may be owned by a surface or volume.

Hex

A hex is a hexahedral element (six connected faces). A hex is owned by the enclosing volume.

Mesh Generation Order of Proccession

The mesh for any given geometry is usually generated hierarchically. For example, if the user issues a command to mesh a volume, first its vertices are meshed with nodes, then curves are meshed with edges, then surfaces are meshed with faces, and finally the volume is meshed with hexes. Vertex meshing is of course trivial and thus the user is given no control over this process. However, curve, surface, and volume meshing can be directly controlled by the user. The **scheme** command specifies the meshing algorithm which will be used in meshing each of these geometric entities and the block command specifies the type of elements which will be generated by the meshing algorithm.

▼ Mesh Attributes

Each geometric entity has mesh attributes which specify information such as meshing scheme, mesh density, mesh distribution (equal or biased), and element type. Unless otherwise specified by the user (see page 94), the default meshing attributes listed in Table 5-1 will be used.:

Table 5-1 Default Meshing Attributes

<i>Geometric Entity</i>	<i>Default Attributes</i>		
	<i>Scheme</i>	<i>Element Type</i>	<i>Intervals</i>
Curve	Equal	2-node Bar	1
Surface	Map	4-node Quadrilateral	—
Volume	Map	8-node Hexahedral	—

Meshing Schemes

The mesh scheme attribute specifies the meshing algorithm that will be used to generate the mesh on each geometric entity. Note that the meshing scheme can be specified independently for curves, surfaces, and volumes; however, the meshing schemes for all surfaces on a volume must be compatible with the meshing scheme specified for that volume. For example, the Project, Translate, and Rotate volume meshing schemes require that some of the volume

surfaces be meshed using the Mapping scheme. The currently supported meshing schemes are listed in Table 5-2.

Table 5-2 Valid Meshing Schemes for Curves, Surfaces, and Volumes

<i>Scheme</i>	<i>Description</i>
Curve Meshing Schemes	
Equal	linear distribution of nodes along a curve based on arc length of the curve (default)
Biased	the distribution of nodes along a curve by biasing the nodal positions toward one of the curve ends given a growth factor
Featuresize	nodes on the curve are spaced proportional to the smallest straight-line distance to another facet of geometry (vertex, curve, or surface) that doesn't contain the curve or one of the curve's vertices
Surface Meshing Schemes	
Map	standard surface mapping transformation [7] (default)
SubMap	mesh-based auto-decomposition of surface into mappable subregions to produce overall regular mapped mesh
TriMap	generate triangular elements at sharp corners or specified vertices and mesh the remaining surface using the standard mapping transformations
Pave	advancing front algorithm for general surfaces including those with holes [1]
TriPave	generate triangular elements at sharp corners or specified vertices and mesh the remaining surface using the paving algorithm
Triangle	meshing primitive for three-sided regions
Circle	meshing primitive for "circular" regions with graded boundary
Volume Meshing Schemes	
Map	standard volumetric mapping transformations [7] (default)
SubMap	mesh-based auto-decomposition of volume into mappable sub-volumes to produce a regular mapped mesh
Project	2&1/2D Sweeping Algorithm—general purpose sweep path, accepts draft angles
Translate	2&1/2D Sweeping Algorithm—along a vector
Rotate	2&1/2D Sweeping Algorithm—about a central axis, requires non-zero inner radius
Plaster	Research algorithm for automatic hexahedral volume meshing
Weave	Research algorithm for automatic hexahedral volume meshing

Automatic Scheme Selection

For Volume and Surface geometries the user may allow CUBIT to automatically select the meshing scheme. Automatic selection is based on several constraints, some of which are controllable by the user. The algorithms to select meshing schemes will use topological and

geometric data to select the best meshing tool. The command to invoke automatic scheme selection is:

<Surface|Volume|Group> <range> Scheme Auto

Specifically for surface meshing, mesh density (see page 86) specifications will affect the scheme designation. For this reason it is recommended that the user specify the mesh density before calling automatic scheme selection. If the user later chooses to change the interval assignment, it may be necessary to call scheme selection again. For example, if the user assigns a square surface to have 4 intervals along each curve, scheme selection will choose the surface mapping algorithm. However if the user designates opposite curves to have different intervals, scheme selection will choose paving, since the mapping tool can't do element transitioning. In cases where a general interval size for a surface or volume is specified and then changed, scheme selection will not change. For example, if the user specified an interval size of 1.0 a square 10X10 surface, scheme selection will choose mapping. If the user changes the interval size to 2.0, mapping will still be chosen as the meshing scheme from scheme selection. If a mesh density is not specified for a surface, a size based on the smallest curve on the surface will be selected.

When automatic scheme selection is called for a volume, surface scheme selection is invoked on the surfaces of the given volume. Automatic volume scheme selection depends heavily on the selection of the surface schemes so these must be selected first, this is done automatically. Mesh density selections should also be specified before automatic volume scheme selection is invoked due to the relationship of surface and volume scheme assignment.

Surface scheme selection will choose between **Pave**, **Submap**, **Triangle**, and **Map** meshing schemes. Volume scheme selection chooses between **Map**, **Submap** and **Project** meshing schemes.

Surface scheme selection will always result in selecting a meshing scheme due to the existence of the paving algorithm, a general surface meshing tool. Volume scheme selection is limited however to selecting schemes for 2.5D geometries, with additional tool limitations (i.e. project can currently only sweep from several sources to a single target, *not* multiple targets). If volume scheme selection is unable to select a meshing scheme, the mesh scheme will remain as the default.

Volume scheme selection can fail to select a meshing scheme for two reasons. First, the volume is not 2.5D and further decomposition (see "Geometry Decomposition" on page 75) of the model is needed. This can be useful for guiding decomposition and reduce the need for the user to visually inspect each volume in the model and save his/her resources for focusing on areas that are unmeshable.

Second, volume scheme selection may fail due to improper surface scheme selection. Volume schemes such as **Map**, **Submap**, and **Project** require certain surface meshing schemes, as mentioned previously. Commonly, a linking surface's meshing scheme on a 2.5D volume will be selected as **Pave** or **Triangle** incorrectly due to the "fuzziness" of the surface's boundary. A fuzzy surface boundary is where the surface is not by geometric description "blocky" and more general in shape. The user can over come this by several methods. First, the user can manually set the surface scheme for the "fuzzy" surface and retry volume scheme selection. Second, the user can manually set the "vertex types" for the surface (see page 91). Third, the user can increase the angle tolerance for determining "fuzziness." The command to change scheme selection's angle tolerances is:

[set] Scheme Auto Fuzzy Tolerance {value} (value in degrees)

The acceptable range of values is between 0 and 360 degrees. If the user enters 360 degrees as the fuzzy tolerance, no fuzzy tolerance checks will be calculated, and in general mapping and submapping will be chosen more often. If the user enters 0 degrees, only surfaces that are “blocky” will be selected to be mapped or submapped, and in general paving will be chosen more often.

In general automatic scheme selection reduces the amount of user input. Generally if the user knows the model consists of 2.5D meshable volumes, three commands to generate a mesh after importing or creating the model are needed. They are:

volume all size <value>
volume all scheme auto
mesh volume all

The following non-trivial, academic model was meshed using these three commands, part of the model is not shown to reveal the internal structure of the model.

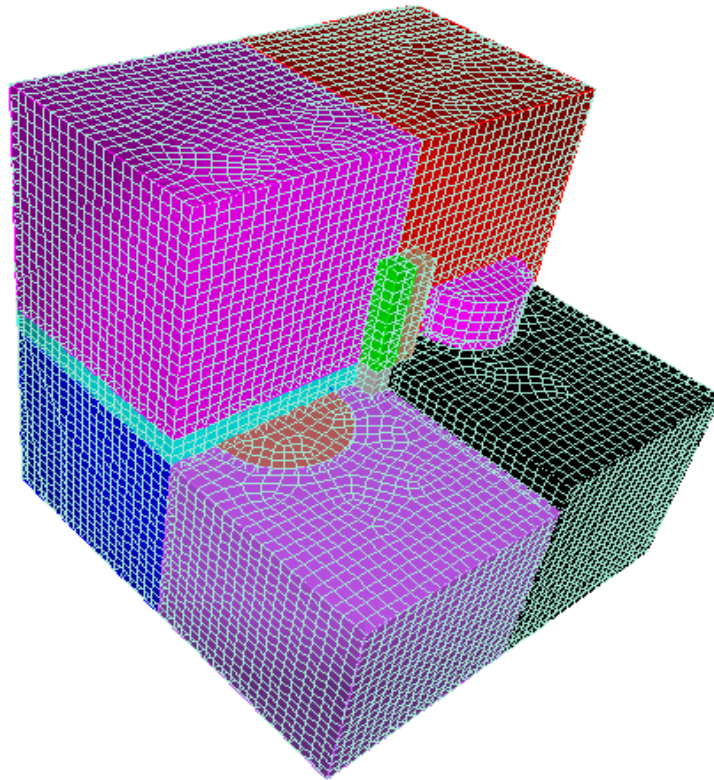


Figure 5-1 Model Meshed Using Automatic Scheme Selection

Scheme Firmness

Meshing schemes may be selected through three different approaches. They are: default settings, automatic scheme selection, and user specification. The user may query the model about a surface or volume to determine what approach was used to set the scheme (see “Model Information” on page 52). Some algorithms, such as automatic scheme selection depend on the “firmness” of the scheme or how the scheme was selected. For instance, if a surface’s meshing scheme was selected by the user, scheme selection will respect that decision and consider the

meshing scheme to be **HARD** set, and will not alter it. If the scheme was set internally by CUBIT, either through scheme selection itself or some other means, it will be treated as **SOFT**, meaning it will only alter the scheme if any parameters (vertex types, or interval settings) have changed. If the scheme is the **DEFAULT**, scheme selection will decide the most appropriate scheme. The user is given ultimate control over these settings by the command:

<Surface|Volume> <range> Scheme {Default|Soft|Hard}

This may be useful if the user is working on several different areas in the model. Once she/he is satisfied with an area's scheme selection and doesn't want it to change, the firmness command can be given to hard set the schemes in that area. Or if some surfaces were hard set by the user, and the user now wants to set them through automatic scheme selection then she/he may change the surface's scheme firmness to soft or default.

Mesh Density Specification

Interval settings control the discretization density of the generated mesh. The number of intervals, or discretizations, can be set on a body, volume, surface, or curve. A related setting, the interval size, specifies the *length* of element edges on a curve, rather than the number of intervals. An advantage of the interval size option is that consistent element sizes can be specified throughout the mesh. Manual interval setting commands are described in "Curve Meshing" on page 87.

Appropriate interval assignment is critical to produce high quality meshes using the map, submap, and triangle surface meshing schemes. When the scheme designation for a surface is *Map*, *Submap*, or *Triangle*, or the scheme designation for a volume is *Map*, *Submap*, *Project*, *Translate*, or *Rotate* automated interval assignment tasks are performed prior to meshing the surface or volume. Use the following command automatically assign compatible intervals to a collection of entities:

Match Intervals entities

Here entities can be any mixed collection of groups, bodies, volumes, surfaces and curves.

The *Map*, *Submap*, *Project*, *Translate*, *Weave*, *Plaster* and *Rotate* volume meshing algorithms automatically execute the automated interval assignment algorithm and do not require explicitly issuing the match intervals command unless it is desired to match intervals on a group of surfaces or group of volumes simultaneously. Both the volume meshing commands and the manual (match intervals) command formulate a list of surfaces that are to have automated interval assignment tasks performed (based on meshing scheme). The list of surfaces is then sent to the automated tool which determines dependencies between intervals on curves and assigns compatible intervals according to the meshing scheme.

The automated interval assignment algorithm calculates a solution as close as possible locally to the initial specified intervals settings, while satisfying global dependencies and compatibility constraints. When the actual *number* of intervals is specifically set by an explicit "curve <range> intervals <value>" command, the intervals are a constraint and designated hard set" and not adjusted. Otherwise, or when the interval *size* is specified, it is assumed that the user is specifying an approximate number of intervals to be placed on a curve rather than an absolute number. These intervals are goals and are designated "soft set" and can be adjusted. If the number of intervals on a curve is not even implicitly specified by the user, it is assumed that the user has no preference for the number of intervals. The curve has no goal and is designated "default", and matching may set intervals arbitrarily. Adjustments to "soft set" curves are minimized on a local basis by the automated interval assignment algorithm while satisfying

dependencies and compatibility constraints. The user may also explicitly set the level of firmness (default, soft, or hard) for the curves of an entity, overriding any previous settings. The user may also require an even number of intervals on a curve.

The automated interval assignment algorithm finds one good feasible solution from among the possibly infinite number of interval solutions. However, if many curves are hard-set or already meshed, there may be no solution. Also, a solution might not exist due to the way the local selections of corners and sides of surfaces interact globally. To improve the chances of finding a solution, it is suggested that curves be soft-set whenever that is acceptable to the user. The user may attempt to find and fix the global corner and side picking problems by specifying vertex types, but this can be a difficult and tedious process. A simpler solution is for the user to make the constraints easier by selecting scheme pave instead of the structured algorithms for some surfaces.

The following sections define the constraints that the automated interval assignment algorithm follows for the Map, Submap, and Triangle, and Pave meshing schemes.

Element Types

CUBIT supports several element types, including bars, beams, quadrilaterals, shells, and hexahedrons. Two- and three-node bar and beam elements; four-, eight-, and nine-node quadrilateral and shell elements; and eight-, twenty-, and twenty-seven node hexahedral elements are available. Multiple element types can be used in a single CUBIT model. The **Block** commands are used to set element types. These are described in “Element Block Specification” on page 134. The local element node numbering is as specified in the Exodus II specification and shown in Figure 5-2.

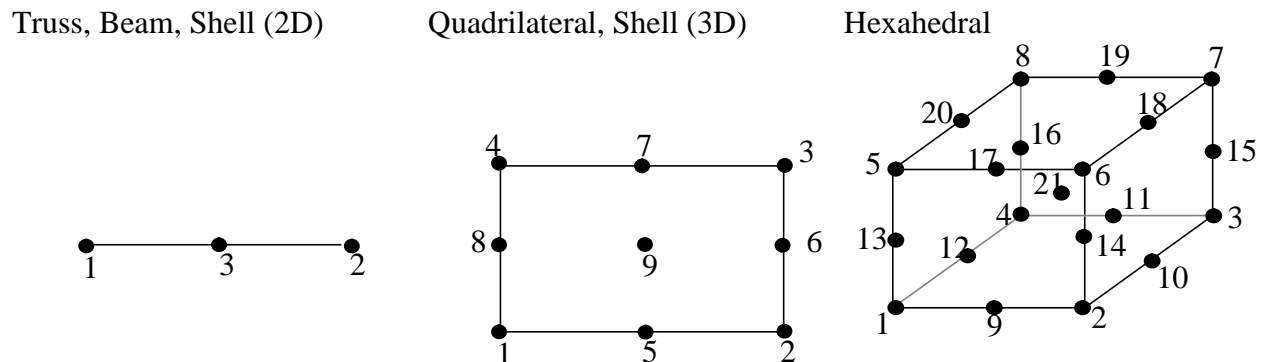


Figure 5-2 Local Node Numbering for CUBIT Element Types



Note: The type of elements to be generated on a geometric entity must be specified prior to meshing the geometric entity unless the defaults listed in Table 5-1 are desired.

▼ Curve Meshing

Curve meshing discretizes the curve, creating nodes and edges. During curve meshing the user controls the density of nodes/edges (or intervals) along the curve and the relative spacing or bias of the nodes along the edge.

Interval Firmness and Parity

The *firmness* of intervals may be *hard*, *soft*, or *default*. Hard-set intervals are never adjusted by the meshing algorithms—even when such an adjustment may produce a better mesh. Soft-set intervals are a goal and may be adjusted up or down slightly. Default intervals are set arbitrarily by the meshing and interval matching algorithms, depending on external factors. The number of intervals on a curve will not be adjusted after that curve has been meshed, either explicitly or as the result of meshing a surface or volume containing that curve. Intervals can be changed if the existing mesh is first deleted. Higher-firmness interval setting commands take precedence over lower-firmness commands, so that a hard-set interval is not changed by any subsequent soft-setting commands. Among a firmness class, the last-issued command takes precedence. E.g. if a curve has its intervals hard-set to 10, then a command to set the containing volume's intervals to 3 will have no effect on the curve, but a command to hard-set the curve's intervals to 12 will change the curve's intervals. The explicit commands to change the firmness is:

{curve|surface|volume|body|group} Interval {Default | Soft | Hard}

The user can also constrain the parity of intervals on curves:

{curve|surface|volume|body|group} <id_range> Interval {Even | Odd}

If “even” is specified, then during subsequent interval setting commands and during interval assignment, curves are **forced** to have an even number of intervals. If the current number of intervals is odd, then it is increased by one to be even. If “odd” is specified then intervals **may be either** even or odd. Unless user specified, curves are “odd”. Setting intervals to even is useful in e.g. problems where adjoining faces are paved one by one without global interval assignment.

Standard Node Density

The density of edges along curves is specified by setting the actual number of intervals or by specifying a desired average interval size. The number of intervals or interval size can be explicitly set curve by curve, or implicitly set by specifying the intervals or interval size on a surface or volume containing that edge. For example, setting the intervals for a volume sets the intervals on all curves in that volume. All curves are initially default, and any command that changes intervals (including interval assignment) upgrades the firmness to at least soft. Note that if higher-order elements are being generated, the number of intervals and interval size refer to the edge of an element, not necessarily to the spacing of nodes along that edge.

The commands to specify the number of intervals at the command line are:

{curve|surface|volume|body|group} <range> interval <intervals>

where **range** may be a single integer or a range of integers. Intervals are soft-set, unless “curve” is specified in which case they are hard-set. The following commands also soft-set intervals. Interval size may be specified at the command line using similar commands:

{curve|surface|volume|body|group} <range> size <interval_size>

The user may also specify that the size is the arc length of the smallest curve:

{curve|surface|volume|body|group} <range> size smallest curve

Relative Element Edge Lengths

The relative length of element edges along a curve is specified using the curve scheme. Two curve schemes are currently supported: **equal** and **bias**. The **equal** scheme generates elements

with equal length edges along the curve. The **bias** scheme requires the specification of a *bias factor* which designates a geometric progression of element edge lengths along the edge. For example, a bias factor of 0.9 will make (as much as possible) each edge along the curve 0.9 times the length of the previous edge, starting at the first vertex¹. The default bias factor is 1.0.

The command used to specify the curve scheme at the command line is:

curve <range> scheme {equal | bias} factor <factor>

The **factor** must be provided if using the bias scheme. Unless a factor of 1.0 is used, this scheme will result in a curve mesh that has mesh nodes distributed in a geometric progression which will be concentrated toward one end of the curve using the progression calculation described above. If a curve meshed with the bias scheme needs to have its nodes distributed towards the opposite end, it can be easily edited using the **reversebias** command. The command used to reverse the bias at the command line is:

curve <range> reversebias

Reversing the curve bias using this command is equivalent to setting a bias factor equal to the inverse of the original bias factor.

Sizing Function-Based Node Density

The ability to specify the number and location of nodes based on a general field function is also available in CUBIT. With this capability the node locations along a curve can be determined by some field variable (e.g. an error measure). This provides a means of using CUBIT in adaptive analyses. To use this capability, a sizing function must have been read in and associated to the geometry (see “Adaptive Surface Meshing” on page 97 for more information on this process). After a sizing function is made available, the following scheme will mesh the curve adaptively:

curve <range> scheme stride

Featuresize Function Node Density

The user may also automatically bias the mesh from small elements near complicated geometry to large elements near expanses of simple geometry. Meshing a curve with scheme *featuresize* places nodes roughly proportional to the distance from the node to a piece of geometry that is *foreign* to the curve. Foreign means that the geometric entity doesn’t contain the curve, or any of its vertices; that is, the entity’s intersection with the curve is empty. It is known that *featuresize* is a continuous function that varies slowly. *Featuresize* meshing is very automatic and integrated with interval matching. *Featuresize* meshing works well with paving, and in some cases with structured surface-meshing schemes as well.

curve <range> scheme featuresize

If desired the user may specify the exact or goal number of intervals with a *size* or *interval* command, and then the *featuresize* function will be used to space the nodes. Also, the *featuresize* function may be scaled by the user to produce a finer or coarser mesh. The default scaling factor or *density* is 1. Note that higher densities also reduce the slope of the function. A

1. The first vertex of a curve can be determined with the list **curve <id>** command.

density of 2 usually gives a good quality mesh. A density below about 0.5 could produce rapid transitions and poor mesh quality.

curve <curve_id_range> density <density_factor>

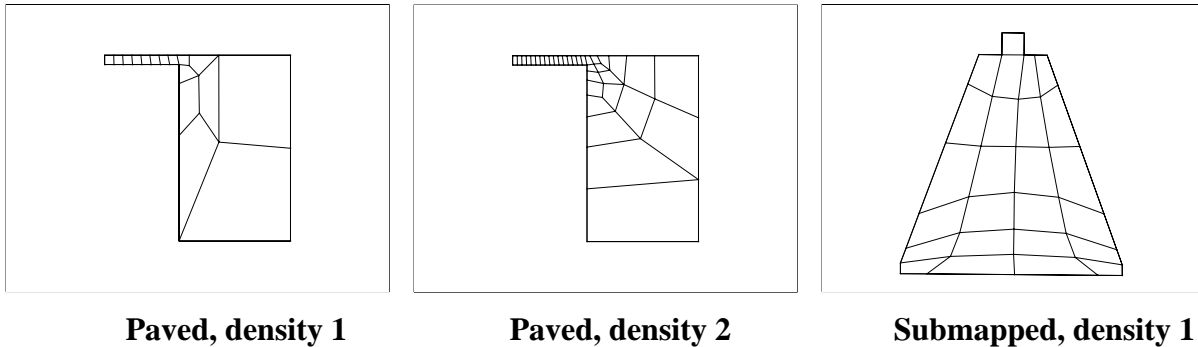


Figure 5-3 Curves meshed with featuresize.

Meshing the Curve

Once the appropriate interval and scheme settings have been made, the curve can be meshed. At the command line, the appropriate command is:

mesh curve <range>

The resulting mesh will be drawn on the screen in the mesh color designated for the curve. Figure 5-4 shows the result of meshing two edges with equal and bias schemes.

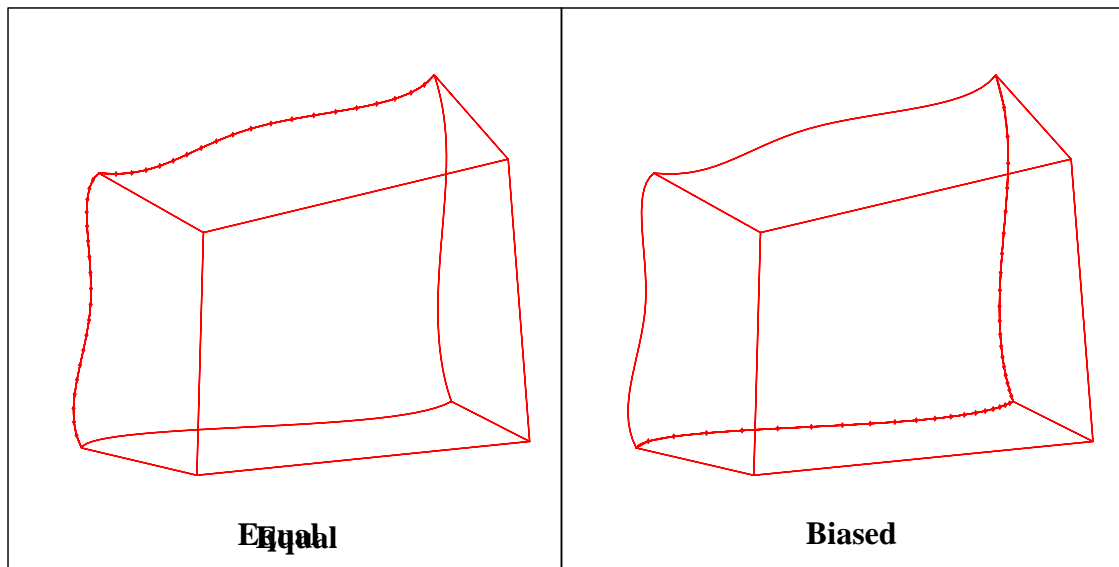


Figure 5-4 Equal and biased curve meshing

▼ Surface Meshing

Surface meshing discretizes a surface into nodes, edges, and faces. When meshing a surface, the bounding curves of the surface are first meshed (if not already meshed). The nodes on those curves are then used as the initial data for the surface meshing. Surface meshing algorithms include mapping, paving, submapping, special versions of mapping and paving which handle sharp corners by inserting triangular elements into the corners before proceeding, primitives, and a technique to apply boundary layers, or rows of aspect-controlled elements, to the surface before closing with the paving algorithm. While not a requirement to generate a well-formed mesh, users have the option to specify how vertices on a surface will be used by some of the surface meshing algorithms (See “Surface Vertex Types” on page 91). A related discussion on the constraints of the mapping and submapping surface schemes is presented to provide the user with background information about which geometries are most appropriate for these meshing schemes (See “Mapping and Submapping Interval Constraints” on page 91).

Surface Vertex Types

Often meshing algorithms, in particular algorithms based on the mapping process, must classify the vertices of a surface or volume to produce a high quality mesh. For example, a surface mapping algorithm must identify the four vertices of the surface that best represent the surface as a rectangle. The submapping, triangle primitive, trimap, and tripave meshing schemes have similar vertex identification needs. Although the surface vertex type can usually be assigned automatically, there are sometimes ambiguous cases or special cases in which the user needs to manually specify the classification of a particular vertex to help improve the resulting mesh. The user may also wish to force a vertex to be of a certain type (see Figure 5-5) to achieve a particular mesh characteristic which would not be produced using the automatic approach. The command

Surface <surface_id> Vertex <vertex_id> Type {end|side|corner|reversal}

Surface <surface_id> Vertex <vertex_id> Type {triangle|notriangle}

is used to manually specify the classification of a particular surface vertex. Note that a vertex may be connected to several surfaces and its classification can be different for each of those surfaces. Figure 5-5 illustrates the vertex angle types. Note that one element will be inserted at an end vertex, two elements at a side vertex, three elements at a corner vertex, and 4 elements at a reversal vertex.

Note: The Surface Vertex Type command does not need to be given in order to mesh a surface, however in some cases, it can improve the quality of the generated mesh.

Mapping and Submapping Interval Constraints

For surfaces with mapping schemes, first the designation of a “*logical rectangle*” fit to the surface is made. The mapping algorithm selects four vertices that will best transform the surface into a logical quadrilateral. These four vertices are chosen as “*logical corners*” and curves falling between these vertices are grouped as a “*logical side*.” In Figure 5-6, the *logical corners* selected by the algorithm are indicated by arrows. Between these vertices the *logical sides* are defined (see Table 5-1).

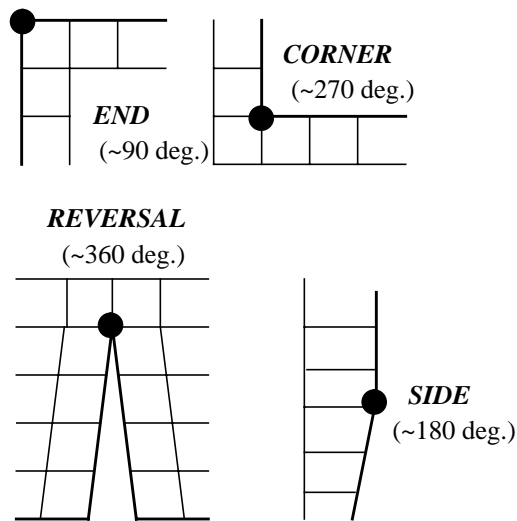


Figure 5-5 Illustration of Angle Types

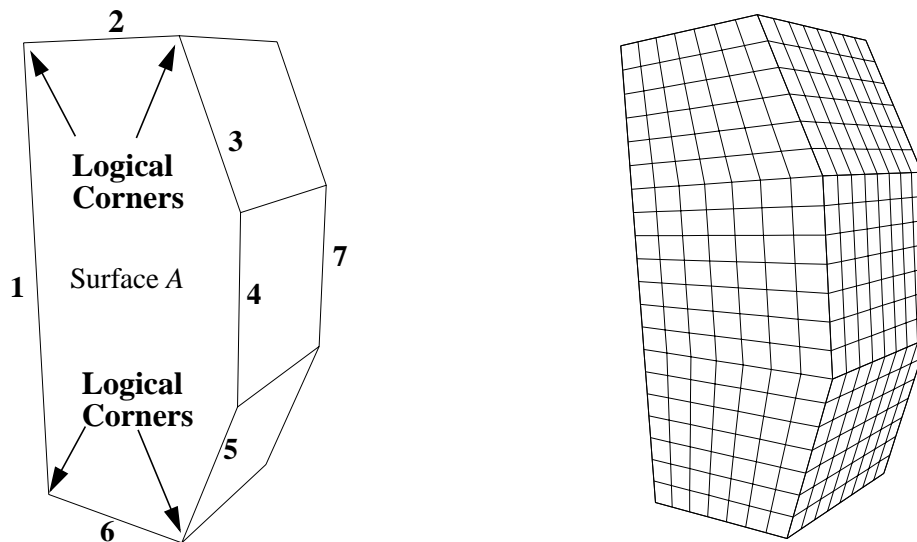


Figure 5-6 Scheme Map Logical Properties

Interval divisions on opposite sides of the logical rectangle are matched to produce the mesh

Table 5-1 Listing of logical sides

Logical Side	Curve Groups
Side 1	Curve 1
Side 2	Curve 2
Side 3	Curve 3, curve 4, curve 5
Side 4	Curve 6

shown in the right portion of Figure 5-6 (i.e. The number of intervals on logical side 1 is equated to the number of intervals on logical side 3).

The process is similar for volume mapping except that a logical hexahedron is formed from eight vertices.

For surfaces with scheme submap the designation of the “*logical rectangle*” is different from that of scheme map rectangle. Curves on the surface are traversed and grouped into “*logical sides*” by a classification of the curves position in a local “i-j” coordinate system. The local “i-j” coordinate system is defined by a traversal of the surface boundary curves. The traversal of the boundary and classification of curves is based on the interior surface angles at each vertex on the surface. Example, curve 1 may be arbitrarily defined in the coordinate system as [+i], a 90 degree turn defines curve 2 as [+j], another 90 degree turn defines curve 3 as [-i], a 270 degree turn defines curve 4 as [+j] and so on. The logical sides are then defined by grouping all curves with the same classification into one side. Therefore all [+i’s] are grouped as one side and all [+j’s] are grouped as another side and so forth. These four sides then define the “logical rectangle” that is used to formulate constraint equations (i.e. side 1 [+i’s] are equated to side 3 [-i’s] and side 2 [+j’s] are equated to side 4 [-j’s]). Figure 5-7 shows one example of this logical

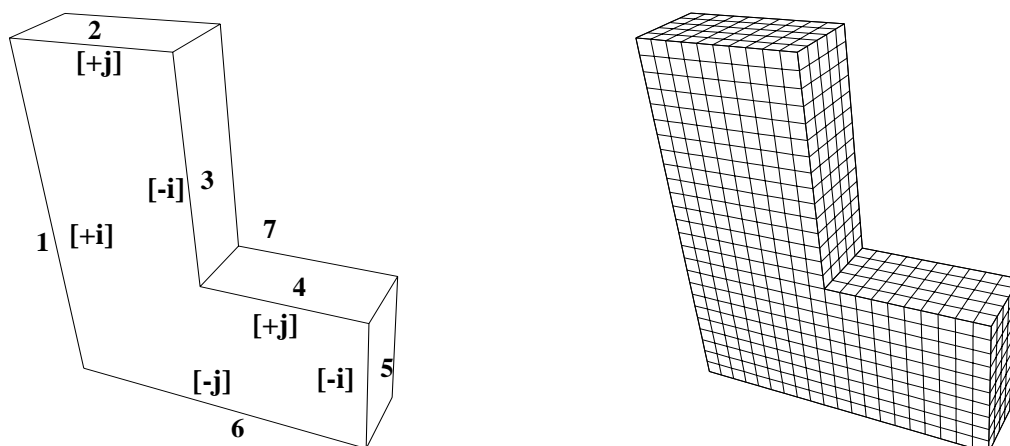


Figure 5-7 Scheme Submap Logical Properties

classification technique.

Scheme Designation

The algorithm to be used for surface meshing is designated as the scheme of the surface. Currently supported surface mesh schemes are **Map** (mapping algorithm), **Pave** (paving algorithm), **Submap** (mapping algorithm with geometry decomposition), **TriMap** (generate triangular elements, map remainder), **TriPave** (generate triangular elements, pave remainder), **Circle** (graded circle primitive), and **Triangle** (triangle primitive). Each of these algorithms are briefly described below. The default scheme for a surface is **Map**. The default scheme may be changed by the command

Surface Default Scheme {map | pave | submap | triangle}

The scheme is set with the command

Surface <range> Scheme {map | pave | submap | triangle | trimap | tripave | dice}

Surface Mapping

The surface mapping capability in CUBIT is based on a standard transfinite interpolation (mapping transformations) [7]. The transformations work robustly and yield high quality meshes in regions with roughly four opposing sides. The surface may have any number of curves defining the sides and still produce a high-quality mesh. Figure 5-8 illustrates a mapped mesh on a NURB surface using two biased and two equal curve meshes on the region's boundary.

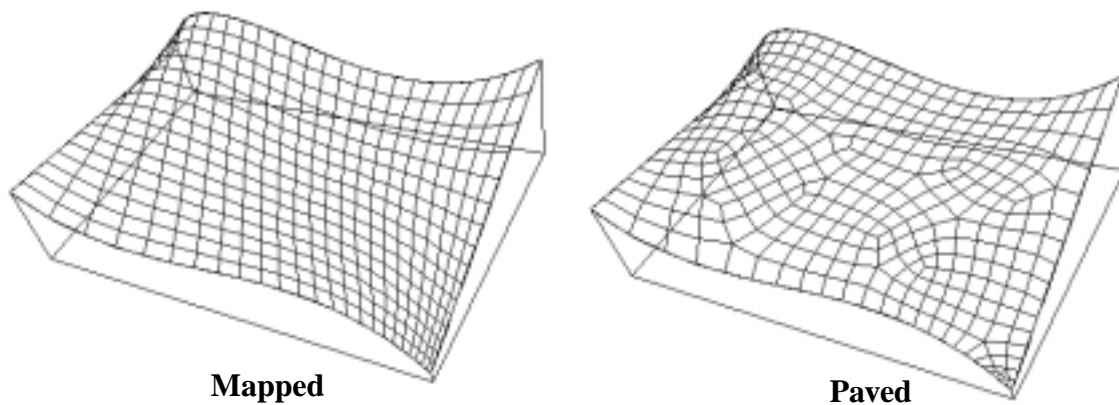


Figure 5-8 Mapped and paved surface meshing

Paving

Paving (see reference [1]) allows the meshing of an arbitrary three-dimensional surface with quadrilateral elements. The paver supports interior holes, arbitrary boundaries, hard lines, zero-width cracks and hard points. It also allows for easy transitions between dissimilar sizes of elements. Figure 5-8 shows the same surface meshed with mapping (left) and paving (right) schemes using the same discretization of the boundary curves.

When meshing a surface geometry with paving, clean-up and smoothing techniques are automatically applied to the paved mesh. These methods improve the regularity and quality of

the surface mesh. Often the type of smoothing method used during this process will affect the resulting mesh quality. By default the paving method uses its own smoothing methods that are not directly callable from CUBIT. Using one of CUBIT's callable smoothing methods in place of the default method will improve mesh quality, sometimes dramatically depending on the surface geometry and specific mesh characteristics. If the user finds that poor element quality is being produced by the paver, switching the smoothing scheme will often help. This is done by the command:

[set] Paver Smooth Method {Default|Smooth Scheme}

Where if the option "Smooth Scheme" is selected, the smoothing scheme specified for the surface will be used in place of the paver's "in house" smoother. See "Mesh Smoothing" on page 124 for more information about the callable smoothing schemes in CUBIT.

After the clean-up and smoothing techniques have been used, the quality of the paved mesh is inspected for extreme element quality namely inverted, flat-sided, concave, or triangular shaped elements. If these elements exist on the surface, a warning is reported to the user identifying the poorly formed element. It should be noted that at this point all possible efforts were exerted to improve these elements, but the result was unsuccessful. The user does however have several options should this occur. The first option is to inspect the mesh and perhaps discover a method to better assign intervals (usually increasing interval size will give the paver a better chance for success). Another method would be to manually decompose the surface to produce an easier surface to mesh. Yet another is to try to move the nodes on the surface manually or collapse faces on the surface mesh. See "Mesh Deletion" on page 126 and "Node and NodeSet Repositioning" on page 127.

In general, creation of these elements is rare and generally limited to thin surfaces with coarse mesh sizes. Efforts are continuing in trying to improve and remove these problems.

Surface Submapping

Submapping is a meshing tool based on the surface mapping capability discussed previously. This tool is suited for mesh generation on surfaces which can be decomposed into mappable subsurfaces. This algorithm uses a limited decomposition method to break the surface into simple mappable regions. Submapping is not limited by the number of corners or reversals in the geometry or by the number of edges. The submap tool, however is best suited for surfaces that are fairly blocky or surfaces that contain interior angles that are close to Cartesian.

After submapping has subdivided the surface and applied the mapped meshing technique mentioned above, the mesh is smoothed to reduce the sharpness of the decomposition. Because the decomposition is mesh based, no geometry is created in the process and the resulting interior mesh can be smoothed. This increases the conformity of the mesh to the surface. Sometimes the smoothing can decrease the quality of the mesh, in this case the following command can turn off the automatic smoothing before meshing:

Surface <id> SubMap Smooth <on|off>

An illustration of a mesh produced by the submapping algorithm is shown in Figure 5-9. The left side of this figure shows the topology assumed by the submapping algorithm and the right side shows the resulting mesh and the vertex classifications. An alternative interpretation of the topology is shown in Figure 5-10.

Surface submapping also has the ability to mesh periodic surfaces such as cylinders. The requirement for meshing these surfaces is that the top and bottom of the cylinder must have

Transformed Geometry by
Surface Vertex Type Command

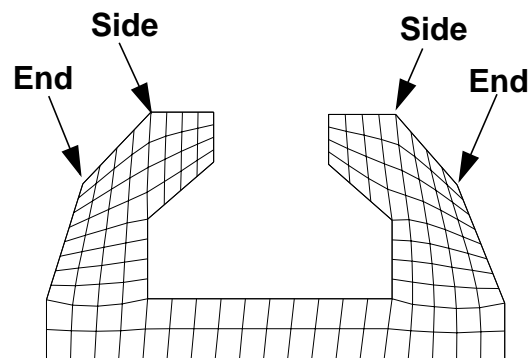
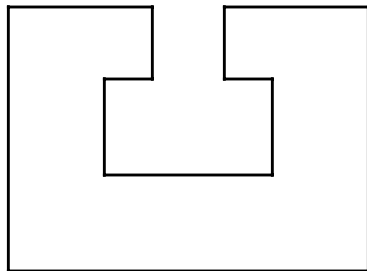


Figure 5-9 Submapping Example

Transformed Geometry by
Surface Vertex Type Command

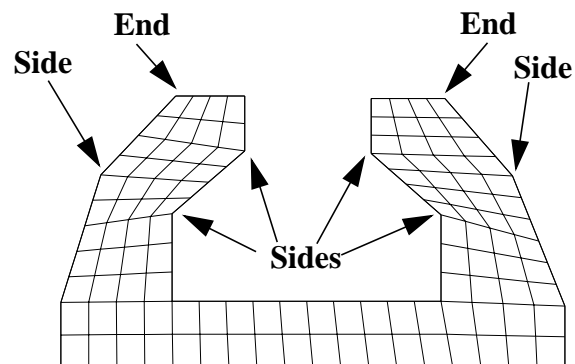
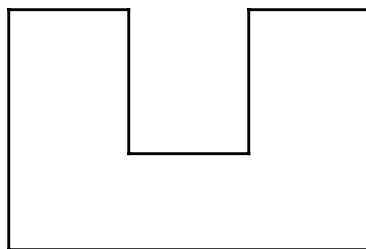


Figure 5-10 Alternate Submapping Topology Interpretation

matching intervals. The intervals between the top and bottom of the cylinder can be set specifically by the command:

Surface <id> Periodic Interval <intervals>

The intervals in the middle can also be set by assigning an interval size to the surface. No special commands need to be given to submap a periodic surface, the algorithm will automatically detect this. Currently, periodic surfaces with interior holes are *not* supported. An example of a periodic surface meshed with submapping is shown in Figure 5-11.

Meshing Primitives

Several basic shapes can be meshed as primitives using an internal decomposition technique to decompose the shape into mappable segments. The triangle and circle primitives are currently the only surface meshing primitive available in CUBIT.

• Triangle Primitive

The **triangle** scheme indicates that the region should be meshed as a triangle. The definition of the triangle is general in that surfaces containing 3 natural corners can often be meshed successfully with this algorithm. For instance, the surface of a sphere octant is handled nicely by the triangle primitive. The algorithm requires that there be at least 6 intervals (2 per side)

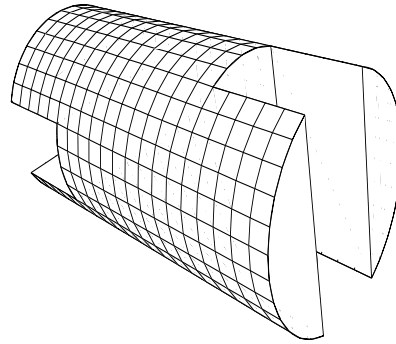


Figure 5-11 Periodic Surface Meshing with Submapping

specified on the curves representing the perimeter of the surface and that the sum of the intervals on any two of the triangle's sides be at least two greater than the number of intervals on the remaining side. Figure 5-12 illustrates a triangle mesh on a 3D surface.

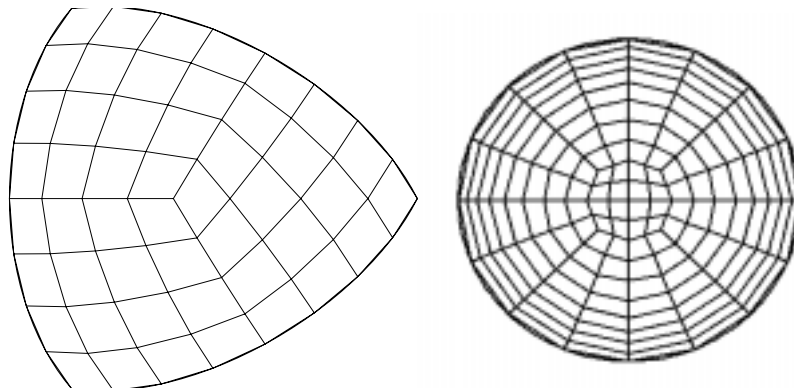


Figure 5-12 Triangle and Circle Primitive Meshes

- **Circle Primitive**

The **Circle** scheme indicates that the region should be meshed as a circle. A “circle” consists of a single bounding curve containing an even number of intervals. Thus, circle can be applied to circles, ellipses, ovals, and regions with “corners” (e.g. polygons). The bounding curve should enclose a convex region. Non-planar bounding loops can also be meshed using the circle primitive provided the surface curvature is not too great. The mesh resembles that obtained via polar coordinates except that the the cells at the “center” are quadrilaterals, not triangles (see Figure 5-12). Radially grading of the mesh may be achieved via the optional [intervals] input parameter. See Ref. XXX for the theory of the circle primitive.

Adaptive Surface Meshing

Adaptive surface meshing in CUBIT produces a function following mesh which sizes elements based on the value of the driving function at the spatial location at which the element is to be placed. Adaptive surface meshing is performed using the paving algorithm in combination with an appropriate sizing function. The types of sizing functions that can be used are curvature,

linear, interval, inverse, super, test, and Exodus-based field function. These are each described in the following paragraphs.

The procedure for adaptively meshing a surface is to designate paving as the mesh scheme for that surface, assign sizing function types, and mesh the surface. The command syntax of these commands is:

Surface < id > Scheme Pave

Surface < id > Sizing Function Type { Curvature | Linear | Interval | Inverse | Super | Test | Exodus }

Mesh Surface <id>

The **Curvature** sizing function determines element size based on the curvature evaluation of a surface at the current location. Two surface curvature values (taken perpendicular to each other) are compared at the location of interest, and the largest is used as the sizing function for the mesh. Figure 5-13 shows a solid with a highly deformed surface which displays rapid change

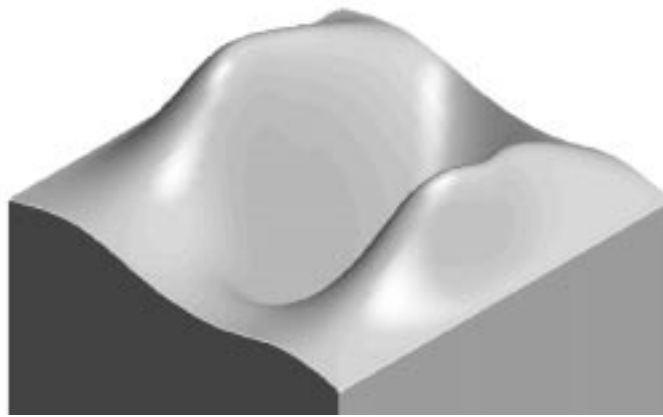


Figure 5-13 NURB solid with high surface curvature change

of surface curvature at several locations. Figure 5-14 depicts a normal paved mesh of this surface using a common size on all bounding curves and no sizing function in the interior. The total number of quadrilateral shell elements for this case is 1988. Figure 5-15 shows a mesh which was generated with the curvature sizing function option. The mesh is graded denser in the regions of quickly changing curvature, such as at the tops of the hills and at the bottom of the valley. Due to the intense interrogation of the underlying geometric modeler which the curvature method relies on, this option can be very computationally expensive.

The **Linear** class of sizing functions determines element size based on a weighted average of edge lengths for mesh edges bounding the surface being meshed. There are several variants of this class of sizing function. The **Linear** function bases edge length at a location on the lengths of edges bounding the surface weighted by their inverse distance from the current location. The result of this weighting is a more gradual change in mesh density during a transition between dense and coarse mesh. Figure 5-16 shows the same NURB surface mesh but with intervals of 34 on two curves and intervals of 16 on the remaining two bounding curves and no sizing function. It can be observed that the mesh progresses more rapidly inward from the coarser meshed curves, which locates the transition region much closer to the finer meshed curves. To combat this, the **Linear** function weights the sizing of new elements such that these transitions

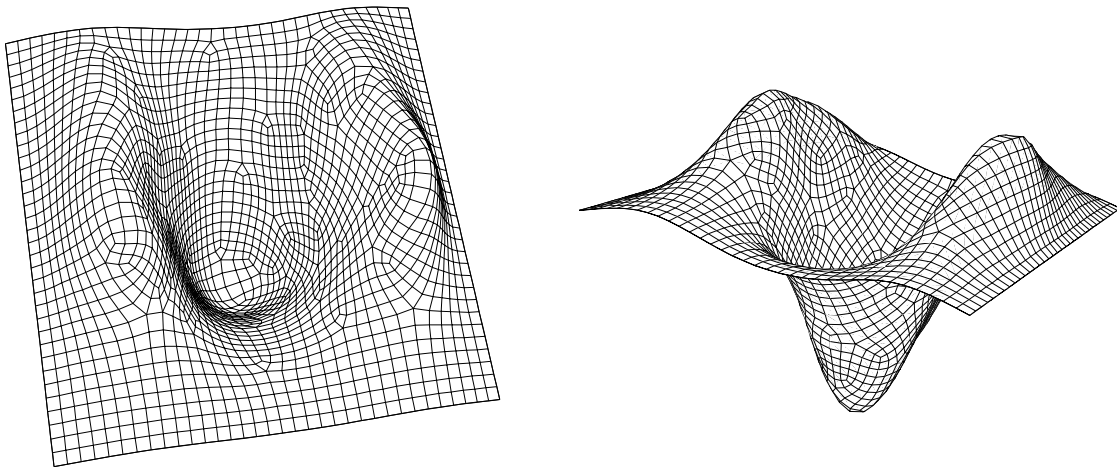


Figure 5-14 NURB mesh with no interior sizing function

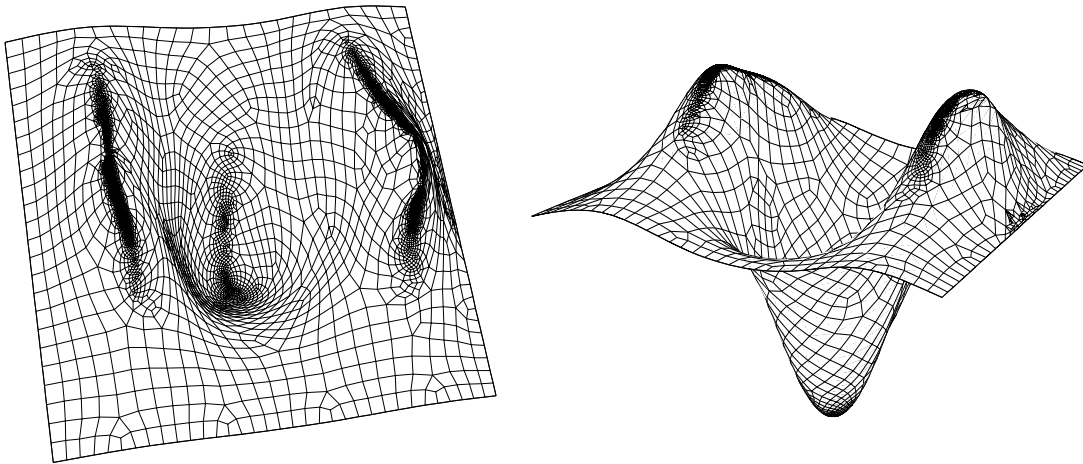


Figure 5-15 NURB mesh with curvature sizing function

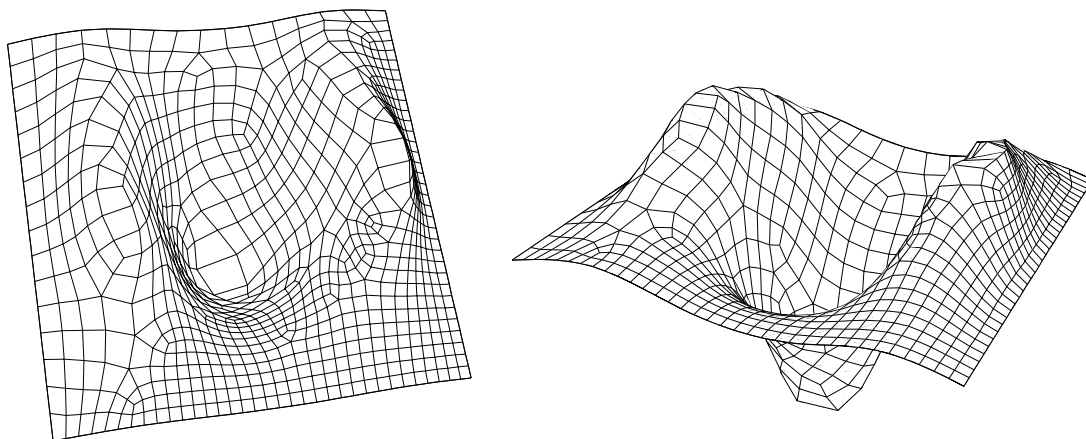


Figure 5-16 NURB mesh with no sizing function, 34 by 16 density

occur slower. Figure 5-17 displays the same NURB geometry with the same bounding curve

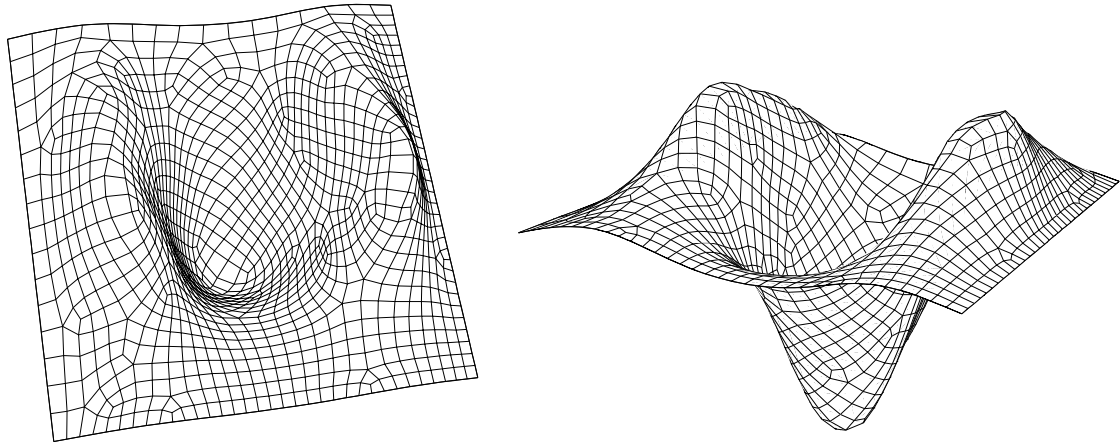


Figure 5-17 NURB mesh with linear sizing function, 34 by 16 density

mesh density using the linear sizing function.

The **Interval** function is similar to the **Linear** function, but bases edge length at a location on the *squared* lengths of edges bounding the surface weighted by their inverse distance from the current location (see Figure 5-18).

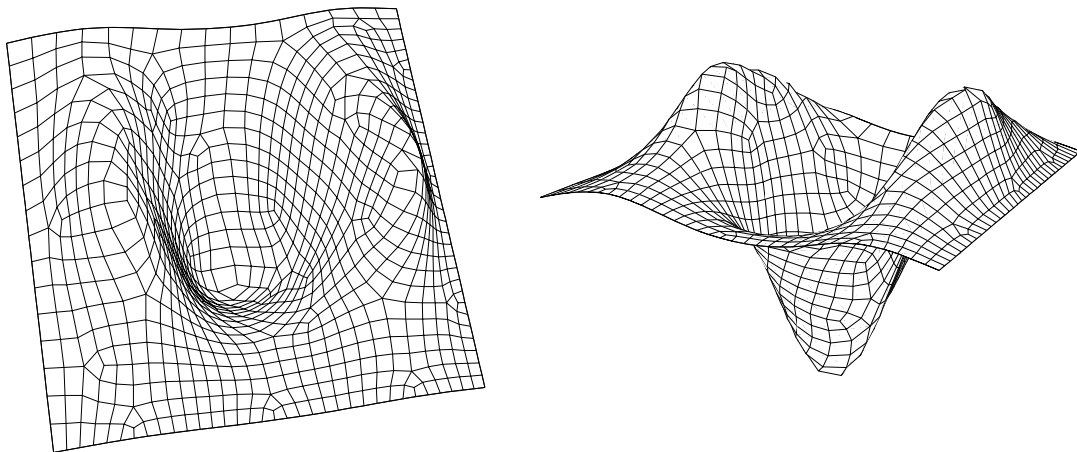


Figure 5-18 NURB mesh with interval sizing function, 34 by 16 density

The **Inverse** function is also similar to the **Linear** function, but this method bases edge length at a location on the *inverse* lengths of edges bounding the surface weighted by their inverse distance from the current location (see Figure 5-19). The difference between the three linear sizing functions is sometimes subtle, but is driven by the geometry being meshed since the influence of these functions is strongly controlled by the number, positioning, and mesh density of the bounding curves relative to the interior surface area.

The **Super** sizing function computes both the **Curvature** and the **Linear** function and takes the smaller value of the two (see Figure 5-20).

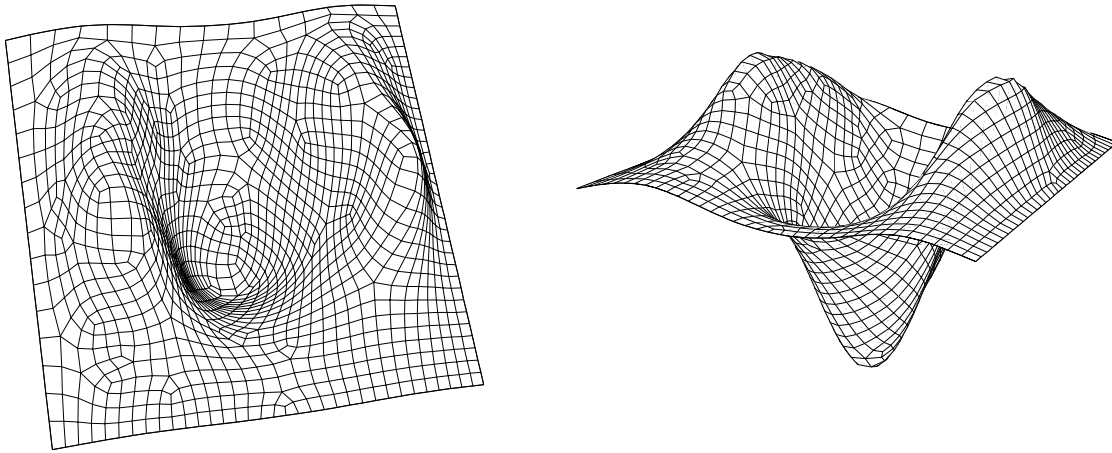


Figure 5-19 NURB mesh with inverse sizing function, 34 by 16 density

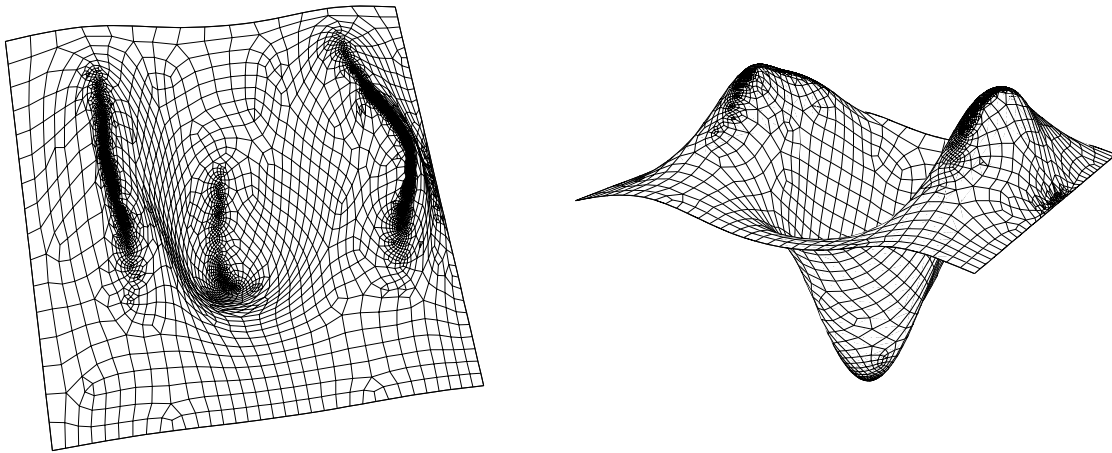


Figure 5-20 NURB mesh with super sizing function, 34 by 16 density

The **Test** sizing function is a hardwired numerical function used to demonstrate the transitional effect of sizing function-based and adaptive paving. The function is a periodic function which is repeated in 50x50 unit intervals on a 2D surface in the first quadrant ($x > 0$, $y > 0$, $z = 0$). A square mesh which was generated using this function is shown in Figure 5-21. Another example is shown in Figure 5-22.

The ability to specify the size of elements based on a general field function is also available in CUBIT. With this capability the desired element size can be determined using a field variable read from a time-dependent variable in an Exodus file. Either node-based or element-based variables can be used. Importing a field function and associating it with a surface, and normalizing that function are done in two separate steps to allow renormalization without having to read the mesh in again. Currently, field functions are imported from element and node-based ExodusII data. Thus, a field function is a time-dependent element variable in an ExodusII file. The mesh block containing the corresponding elements must be imported along with the field function data. For details on the adaptive paving algorithm, see [Ref].

Exodus variable-based adaptive paving is accomplished in CUBIT in several steps:

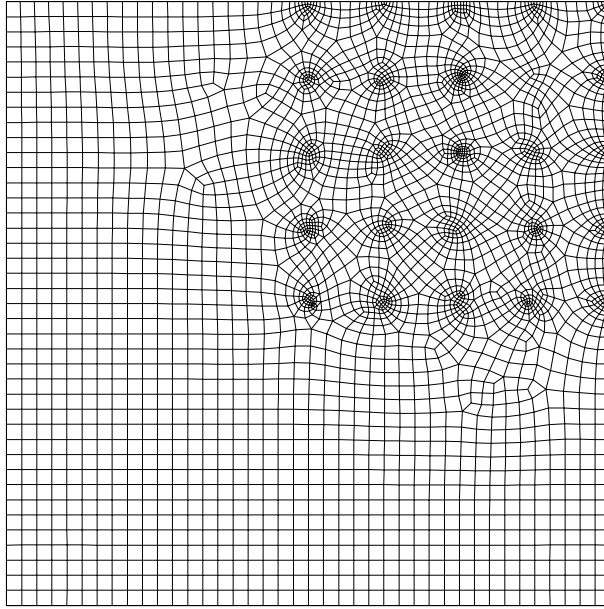


Figure 5-21 Test sizing function mesh for square geometry

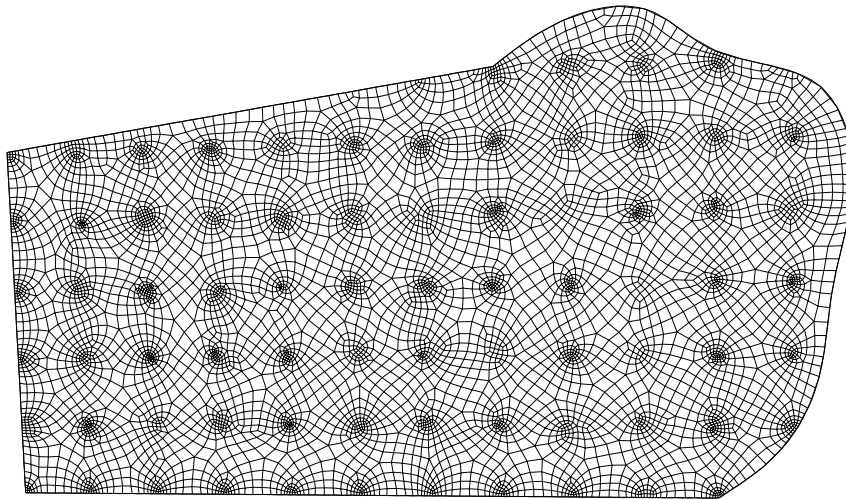


Figure 5-22 Test sizing function for spline geometry

- Surface mesh scheme set to Pave. Bounding curve mesh schemes can also optionally be set to Stride.
- An Exodus mesh and time-dependent variable for that mesh is read into CUBIT.
- The mesh and variable data are associated to geometry.
- The Exodus variable is normalized to give localized size measures, and the surface sizing function type is designated.
- Surface is meshed.

The following command is used to read in a field function and its associated mesh:

**Import Sizing Function '<exodusII_filename>' Block <block_id>
Variable '<variable_name>' Time <time_val> [Deformed]**

where **<block_id>** is the element block to be read, **<variable_name>** is the Exodus time-dependent variable name (either element-based or nodal-based), and **<time_val>** is the problem time at which the data is to be read, the **Deformed** keyword indicates whether deformation has been accounted for on the new model (for information on creating deformed 2D geometry from EXODUSII data, see “Importing EXODUSII Files” on page 70) and needs to be accounted for in the sizing function data. When this command is given, the nodes and elements for that element block are read in and associated to geometry already initialized in CUBIT (for information on associating mesh to geometry, see [ref]). Note that when a sizing function is read in, the mesh is stored in an ExodusMesh object for the corresponding geometry, and therefore the geometry is not considered to be meshed. Also note that if deformation is not being modelled, the geometry to which the mesh is being associated must be in the same state as it was when that mesh was written (see “Mesh Importing and Duplicating” on page 127 for more details on importing meshes).

Once the field function has been read in and assigned to a surface, it can be normalized before being used to generate a mesh. The normalization parameters are specified in the same command that is used to specify the sizing function type for the surface. The syntax of this command is:

**Surface < id > Sizing Function Type Exodus
[Min <min_val> Max <max_val>]**

If normalization parameters are specified, the field function will be normalized so that its range falls between the minimum and maximum values input. Subsequent normalizations operate on the normalized data and not on the original data. If an element-based variable is used for the sizing function, each node is assigned a sizing function that is the average of variables on all elements connected to that node. Nodal variables are used directly.

After the sizing function normalization, the surface can be meshed using the normal meshing command

Mesh Surface <id>

For example, the left image in Figure 5-23 depicts a plastic strain metric which was generated by PRONTO-3D [18], a transient solid dynamics solver, and recorded into an ExodusII data file. When the file is read back into CUBIT, the paving algorithm is driven by the function values at the original node locations, resulting in an adaptively generated mesh [19]. The right image in Figure 5-23 depicts the resulting mesh from this plastic strain objective function.

Boundary Layer Meshing

The Boundary Layer meshing scheme is an algorithm designed to insert rows of elements around the boundary of a surface before meshing the interior. The aspect ratios of these elements can be carefully controlled. This capability is specifically designed for fluid simulations involving a boundary layer. With this tool, the total boundary layer thickness and relative thicknesses of each of the rows can be specified.

A boundary layer is specified as a set of parameters, with a boundary layer ID. This set of parameters is then attached to curve/surface pairs in the geometry. Thus a curve may have a row of boundary layer elements next to it on one of the surfaces it bounds, but not on another. A

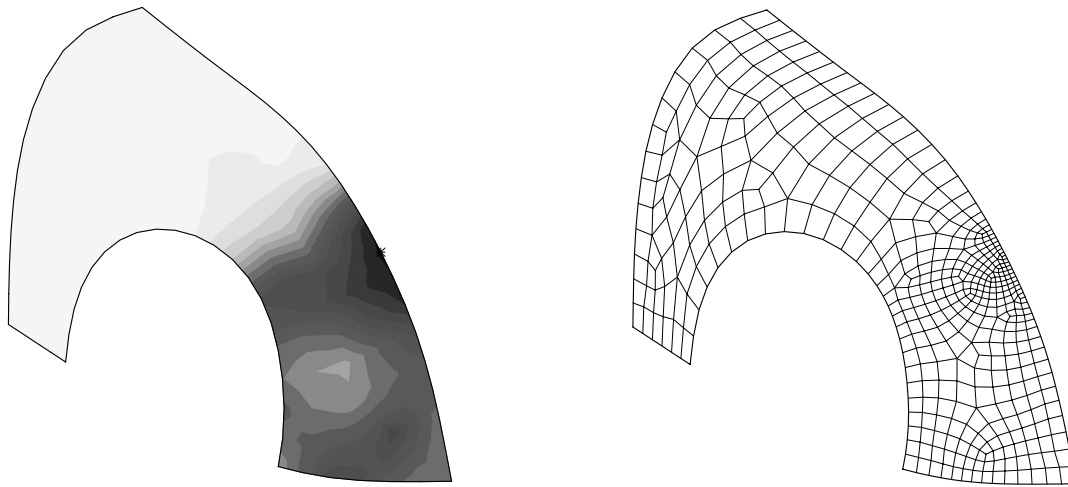


Figure 5-23 Plastic strain metric and the adaptively generated mesh

boundary layer is currently defined using a combination of four of the five possible parameters. These parameters are shown in Figure 5-24.

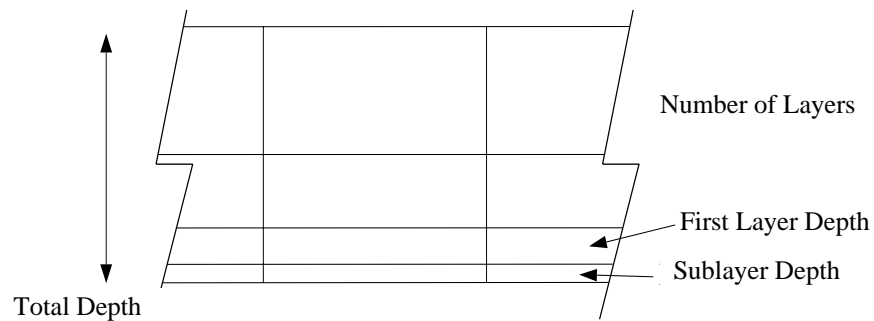


Figure 5-24 Boundary Layer Parameters

- **First Layer Depth.** This parameter specifies the physical depth of the first layer of elements. This is a required parameter.
- **Growth Factor.** This parameter specifies the relative difference of each subsequent layer's depth. For instance, a factor of 1.2 makes each layer 1.2 times the preceding layer's depth. This is an required parameter unless the second layer depth is specified.
- **Total Depth.** This parameter specifies the total depth of all the boundary layers. Either the total depth or the number of layers must be specified.
- **Number of Layers.** This parameter specifies the total number of layers in the boundary layer. Either the number of layers or the total depth must be specified.
- **Sublayer Depth.** This parameter specifies the physical depth of the sublayer of elements. This is an optional parameter. If it is specified a sublayer of elements (not counted in the "number of layers" parameter) is added.

A boundary layer is created with either of the following commands:

BoundaryLayer <range> First [Layer] <depth> Growth [Factor] <growth>
Total Depth <depth> [Sublayer <depth>]

BoundaryLayer <range> First [Layer] <depth> Growth [Factor] <growth>
Layers <count> [Sublayer <depth>]

Note: If the growth factor, total depth, and number of rows are specified together, then the boundary layer definition is overspecified, and the total depth will be ignored.

The boundary layer is attached to curve/surface pairs with the command:

BoundaryLayer <layer_id> Surface <range> Curve <range>

Generating the Surface Mesh

Once the desired scheme has been chosen, and any boundary layers for the surface defined and attached, the surface can be meshed. If the user wishes to receive a different element type than the default (four node quads) for surface meshing, this specification needs to be set prior to creating any surface meshes. The command to mesh a surface is:

mesh surface <range>

The resulting mesh will be drawn on the screen in the mesh color designated for the surface.

▼ Volume Meshing

Volume meshing discretizes the volume into nodes, edges, faces, and hexahedral elements. When meshing a volume, the bounding surfaces of the volume are first meshed (if not already meshed). Available volume meshing algorithms are **mapping**, **submapping**, **project**, **translate**, **rotate**, **plaster**, and **whisker weaving** (**plaster** and **whisker weaving** are currently under development and as such are not recommended for production use). The **StairTool** meshing algorithm is also available, but generates a volume-surrounding mesh rather than a volume-filling mesh.

Scheme Designation

The algorithm to be used for volume meshing is designated as the scheme of the volume. Currently, valid schemes are:

Map Create mesh using mapping transformations.

Submap Create mesh by breaking geometry into several connected regions which are then automatically meshed using mapping transformations.

Project Create mesh by projecting the mesh from one surface to another.

Translate Create mesh by translating the mesh from a source surface along the vector from the source surface to the target surface.

Rotate Create mesh by rotating about an axis from the source surface to the target surface.

Plaster Fill the volume in a free meshing inward approach—currently being researched..

Stair Surround the volume with a non-intersecting, structured shell of quadrilaterals.

Weave Attempt to generate whisker weaving sheets to fill the volume—currently being researched.

Dice Refine an existing mesh by splitting each existing element into a specified number of smaller elements.

Each of these algorithms are briefly described below. The default scheme for a volume is **Map**. The default scheme for volume meshing can be changed by the command:

Volume Default Scheme {map | submap | plaster | weave}

The scheme is set with any of the following commands:

Volume <range> Scheme {map | submap | weave | plaster | pyramid | stair | dice}

Volume <range> Scheme {project | translate | rotate} Source <id> Target <id>

Volume Mapping

The volume mapping capability in CUBIT is based on the same transfinite interpolation mapping transformations [7] which were discussed in the surface meshing section (See “Surface Mapping” on page 94). Volume mapping is designed to work on volumes that can represent a logical cube (they have six logical surfaces and eight logical vertices) as determined by their surface mesh (if the surface mesh contains any irregular nodes, or non-corner nodes that are connected to more or less than four other surface nodes, then mapping transformations cannot be used to generate a volume mesh). There may be more or less actual surfaces, as long as the logical surfaces can be determined. For example the union of two blocks shown in Figure 5-25 contains eight surfaces, but it is easy to see that four side surfaces can be logically

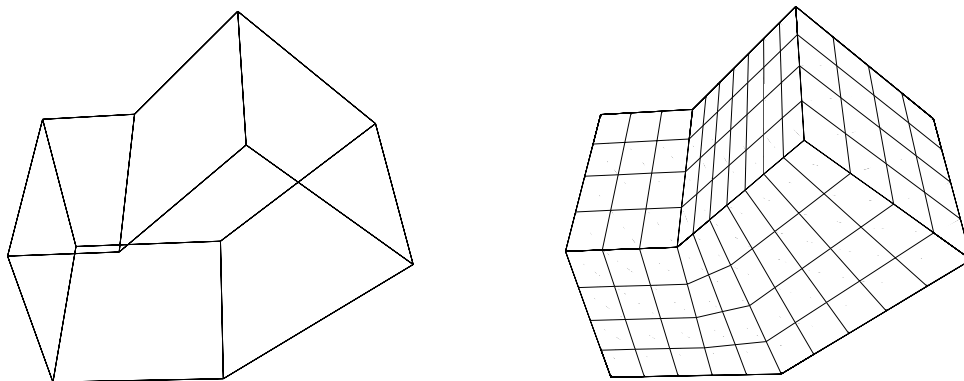


Figure 5-25 Volume mapping of an 8-surfaced volume.

combined to form two surfaces of the logical cube and mapping can be performed successfully. A model which contains less than six surfaces but can be mapped is the quarter cylinder shown in Figure 5-26 which has only five surfaces. However, the cylindrical surface can be logically dissected to form two of the logical surfaces, and this volume can also be meshed successfully. The volume mapper in CUBIT needs no input from the user to determine which of the surfaces need logical dissection and/or combination. The surface mesh, as described below, dictates these choices.

The pattern of the surface mesh will dictate whether a volume can be mapped. On any mappable volume mesh, the surface mesh must contain only 8 trivalent nodes (nodes attached to only 3

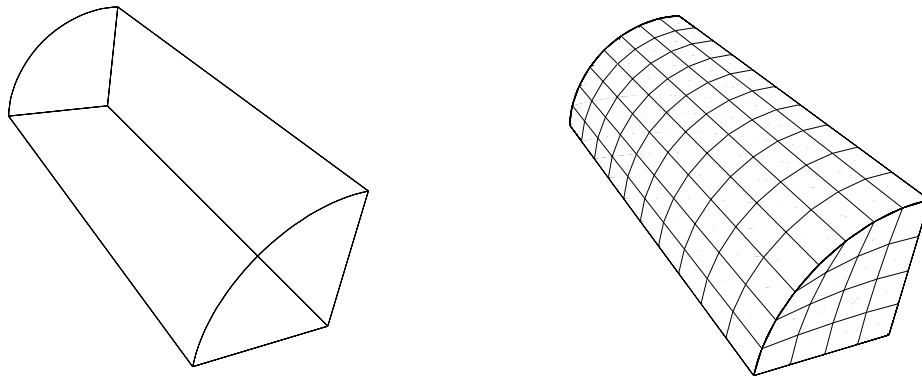


Figure 5-26 Volume mapping of a 5-surfaced volume

quadrilateral faces on the surface). All other nodes must be quadvalent (4 elements attached to the node). These 8 trivalent nodes form the corners of the cube to be mapped. In Figure 5-27

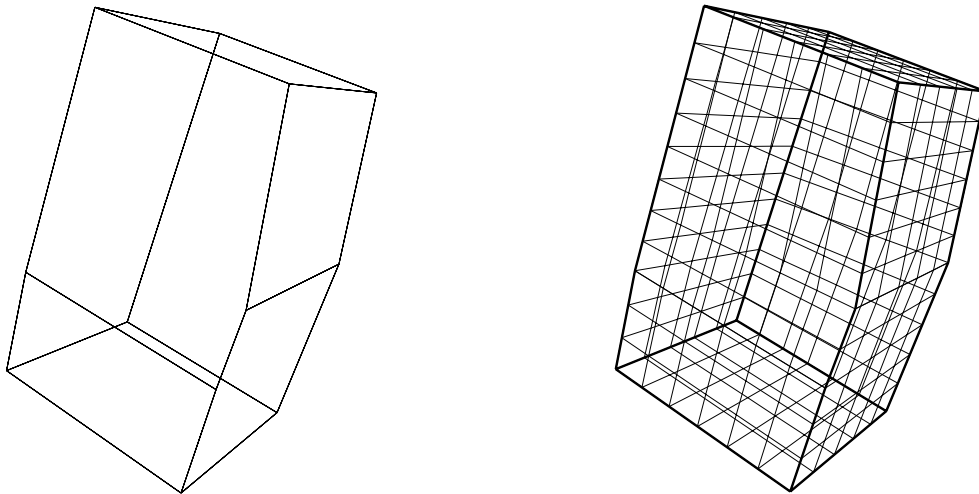


Figure 5-27 Surface mesh of an 8-surfaced volume highlighting the logical edges used for volume mapping.

the surface mesh of an 8-surfaced volume is shown. The logical edges of the surface mesh are highlighted. Notice that not all the geometric edges are used as logical edges for meshing. These logical edges meet at the trivalent nodes of the surface mesh (the corners). This combination of 8 trivalent and the rest quadvalent nodes on the surface can only produce a logical cube. Thus, the user need only insure that the surface mesh has the right characteristics for volume mapping to succeed.

Volume Submapping

Volume submapping is an automated mapping method that uses a mesh based decomposition method to automatically subdivide the volume into mappable sub-volumes. These mappable sub-volumes are defined by mesh boundaries which are then automatically sent to the volume mapper for hexahedral meshing. The sub-volumes are created by meshing virtual surfaces on the interior of the volume. These dissections are created by ‘i-j-k’ space logic (See “Mapping and Submapping Interval Constraints” on page 91 for a discussion of ‘i-j’ space operations). An

example of this can be seen in Figure 5-28, where the volume has been separated into two pieces

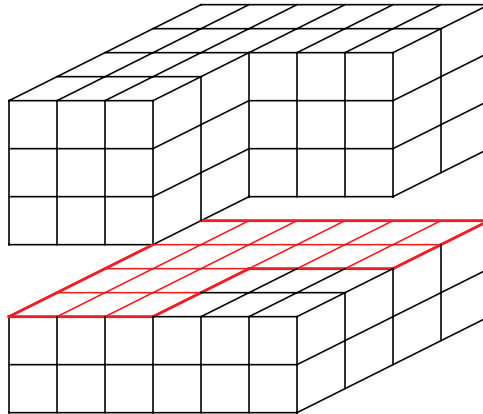


Figure 5-28 Example of internal virtual surface creation

with a virtual surface. The virtual surface will not be visible to the user during or after the meshing process. Volume submapping is limited to geometries that meet the following two criteria: 1) the bounding surfaces have been meshed with surface submapping or mapping, and 2) three, five, and six valent nodes occur only at junctions where surfaces meet.

The command for setting the volume meshing scheme to be submapping is:

volume <volume_id_range> scheme submap

An example of a volume meshed with Volume Submapping is shown in Figure 5-29.

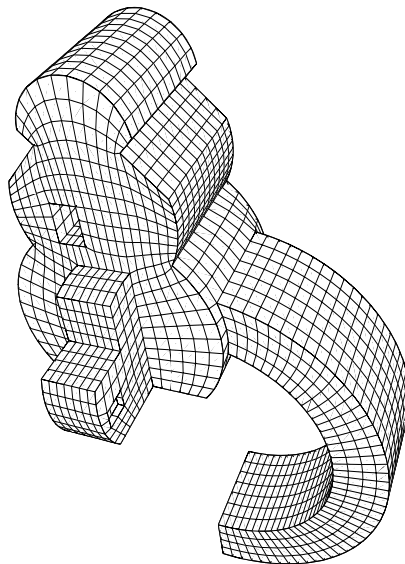


Figure 5-29 Hexahedral mesh generated by volume submapping

Sweeping (Project, Translate, and Rotate)

The CUBIT volume sweeping capability is divided into three algorithms (Project, Translate, and Rotate) which each generate a volume mesh by extruding hexahedrons from a previously

meshed source surface to a topologically similar target surface. Topologically similar includes relationships such as a rectangular surface being extruded into an elliptical surface as long as the two surfaces contain the same number of boundaries or loops¹.

The geometric requirements for a sweeping operation are that the volume be “2 and 1/2 D,” or extrudable. This requirement is typically satisfied if the surfaces linking the source surface and target surface can be meshed with compatible mapping transformations (See “Surface Mapping” on page 94.), where the “compatible” qualifier means that the edges linking the source surface to the target surface have the same number of intervals. The source surface may be meshed using any of the meshing methods described in “Surface Meshing” on page 91. The smoothed topology of the source surface mesh will be reproduced on the target surface unless the target surface is already meshed. In this case, the target surface mesh must have the same topology and connectivity as the source target mesh. It is much more efficient to let the sweeping meshing algorithms mesh the target source face if at all possible.

The procedure for the sweeping volume mesh generation algorithms is as follows: first the attributes (interval settings, element type, etc.) of the volume should be set, and then the surfaces which will act as the source and target must be selected. When the command to mesh the volume is executed, the sweeper will mesh the source surface, and then the linking surfaces. The sweeping algorithms will then project a layer at a time, progressing through the unmeshed volume. The difference between the three sweeping algorithms is the method used to project the nodes and elements from one layer to the next. These details will be discussed in the following sections.

• Project

The **project** sweeping algorithm is a modified version of the plastering hex element projection. The sweep path can be completely general. The nature of the swept region can also be general in that it can contain draft angles and non-symmetric transformations. Figure 5-30 displays swept meshes involving mapped and paved source surfaces. The project algorithm can also

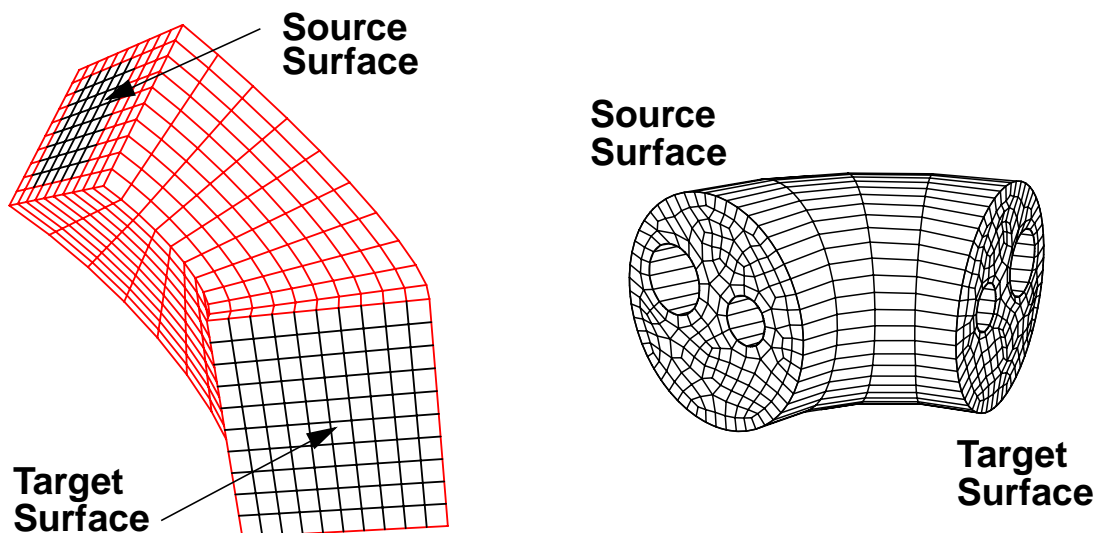


Figure 5-30 Project Volume Meshing

1. The number of loops on a surface refers to the number of boundaries it has. A surface always has at least one boundary, the set of curves which bound it externally. Some surfaces also have internal boundaries, or loops, in the form of holes.

handle multiple surfaces linking the source surface and the target surfaces. An example of this is shown in Figure 5-31. Note that for the multiple surface meshing case, the interval requirement is that the total number of intervals along each multiple edge path from the source surface to the target surface must be the same for each path.

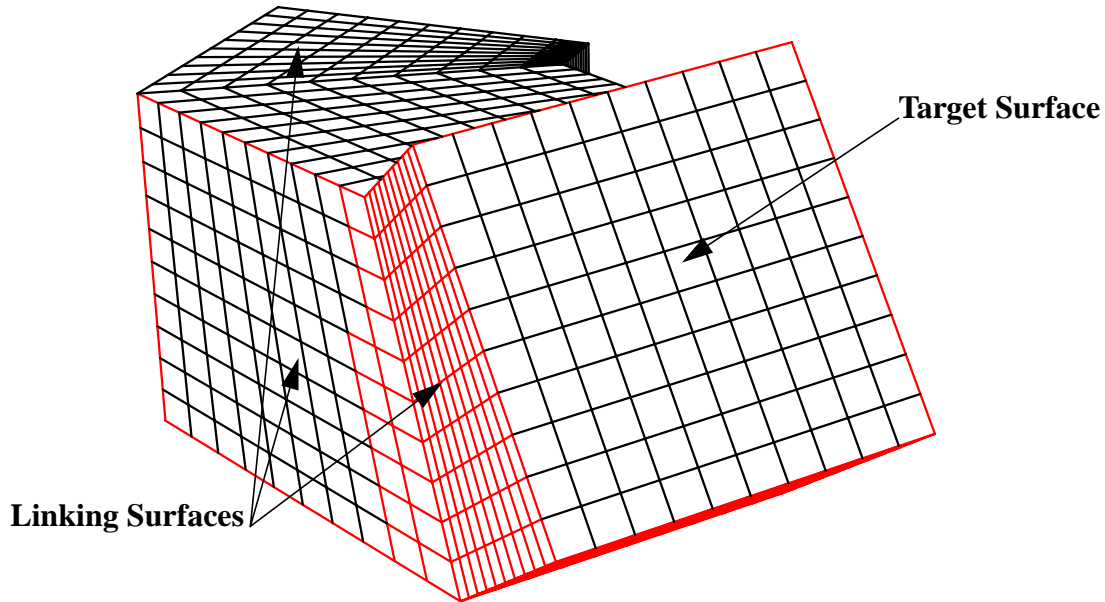


Figure 5-31 Multiple Surface Project Volume Meshing

The project algorithm proceeds by determining a “projection node” for each node on the boundary of the current “layer.” A node’s projection node is the node directly “above” (in the sense of up being closer to the target surface than the source surface). An approximate planar surface is then generated through these nodes. For each node interior to the boundary of the volume, an average “projection vector” is calculated by determining which neighboring nodes have existing projection nodes and averaging the vector from these nodes to their projection nodes. The interior nodes are then calculated by projecting from the interior nodes along the average projection vector to the approximate planar surface calculated earlier. The interior nodes are ordered in a manner to maximize the number of neighboring nodes with existing projection nodes. This process is repeated for each interior node on the current layer. After all projection nodes have been created, a new layer of hexes is created and smoothed. The process then repeats for the next layer.

If the approximate planar surface does not closely match the surface defined by the boundary projection nodes, the interior projection nodes are created simply by projecting along the average projection vector; the intersection with the planar surface is not calculated.

The project algorithm is very general in that it can create a mesh on almost any extrudable volume; however, this generality has some disadvantages in that it does not use any global information about the actual generation of the volume. It simply projects a layer at a time in moving from the source surface to the target surface. Because of this, and the smoothing that is done after each layer is created, features that are present in the source surface mesh sometimes tend to get smoothed out or smeared by the time the mesh reaches the target surface mesh. The project algorithm is also slower and requires more memory than the translate and rotate algorithms since it must calculate a local projection for each node and maintain the information required by the smoothing algorithms.

• **Translate**

The **translate** sweeping algorithm is a more restricted version of the **project** algorithm. It is used when the source surface and target surface have exactly the same geometry and are parallel. If it is possible to translate the source surface along a vector and have it completely overlay the target surface, this algorithm can be used.

The **translate** algorithm proceeds by calculating the vector from the source surface to the target surface. The thickness of each layer is then calculated as the distance from a boundary node on that layer to that node's projection node. This distance is the same for each node on the current layer since the source surface and the target surface (and therefore, each layer) are parallel. Each interior node is then projected that thickness along the vector. This process is repeated for each layer in the volume. No smoothing is performed on the generated volume mesh.

• **Rotate**

The **rotate** sweeping algorithm is also a restricted version of the **project** algorithm. It is used when the source surface and target surface are exactly the same and are connected by a conic or toroidal surface. If it is possible to rotate the source surface about a single axis and have it completely overlay the target surface, this algorithm can be used. This algorithm cannot be used if the rotation axis contacts either the source surface or the target surface, that is, there must be a hole through the center of the generated mesh.

The **rotate** algorithm proceeds by calculating the axis of rotation from the source surface to the target surface. The thickness of each layer is calculated from the amount of rotation from a boundary node on that layer to that node's projection node. This rotational distance is the same for each node on the current layer. This process is repeated for each layer in the volume. No smoothing is performed on the generated volume mesh.

Plastering

Plastering uses the discretized surface and begins to lay elements into the interior of the volume. This continues until the volume fills, with adjustments made to the exterior surface mesh as deemed necessary. This algorithm is currently under development and not suggested for use although it may be tested if desired. It should currently perform well for blocky structures where the surface mesh will form a valid boundary for an interior hex mesh. Some examples of these structures are shown in Figure 5-32. These structures allow very straightforward hex element connectivity and do not contain any irregular nodes (nodes that are shared by other than four element edges in a given layer).

The command for setting the volume meshing scheme to be plastering is:

volume <volume_id_range> scheme plaster

A partial mesh can be created instead of meshing the complete geometry. The number of hex elements or the number of completed layers inward can be specified at the command line when giving the mesh command by using the following syntax:

mesh volume <volume_id_range> hexes <number_hexes>

or

mesh volume <volume_id_range> layers <number_layers>

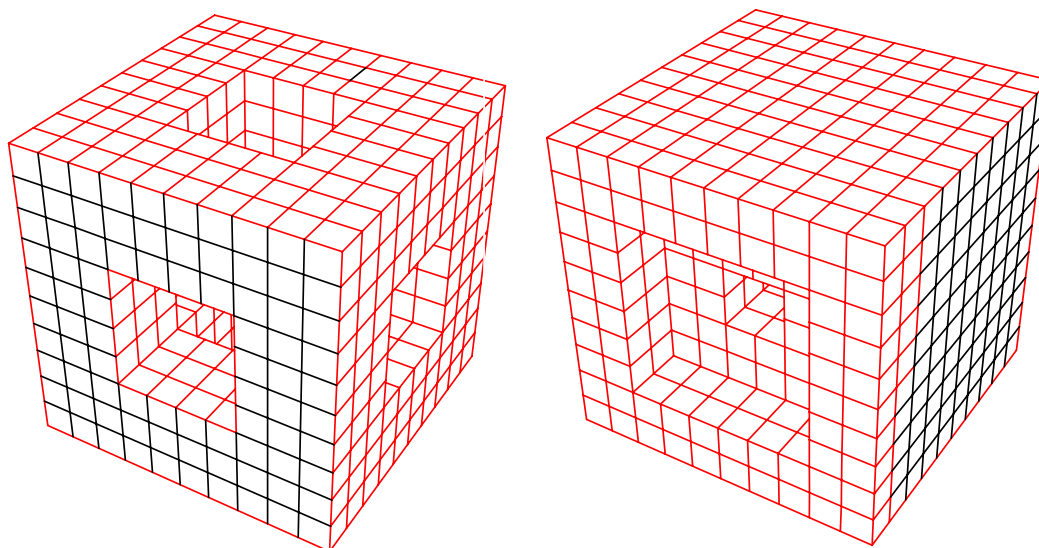


Figure 5-32 Plastering Examples

Stair Tool

The StairTool meshing algorithm surrounds a volume with a non-intersecting, structured, all-quadrilateral shell mesh. The structured quadrilateral shell mesh is axis-aligned, with number of intervals in each coordinate direction user-specified. This mesh is computed using a ray-firing technique combined with face-body intersection checking. StairTool meshes are useful for finite volume methods with applications in electromagnetic and radar cross section modeling.

The StairTool creates a mesh which does not actually lie on a volume; in fact, after stair-meshing a volume, one could conceivably mesh the surfaces and interior of that volume as well. Therefore, StairTool actually creates another body, which is used to hold the mesh. This body is sized to be slightly larger than the bounding box of the original volume, and can be thought of as containing a structured mesh. The StairTool algorithm computes which hexes in this mesh intersect or lie inside the volume, and marks those hexes. The outside shell of these hexes defines the non-intersecting, structured shell mesh output by the StairTool.

The output from the StairTool is stored as a list of quadrilateral elements on the volume created by the StairTool. To output this set of quadrilaterals to an ExodusII file, a new element block must be created with an element type of SHELL4. This block is automatically created by the StairTool, and by default is given an id identical to the new volume's id. This block id can be set by the user.

The following commands are used to control the StairTool meshing algorithm:

body <body_id_range> stair interval { [x <int>] [y <int>] [z <int>] | int }

Sets the number of intervals in the x, y and/or z directions, or, alternatively, sets the number of intervals in all directions. The brick volume surrounding the original volume will have the specified number of intervals in each direction. This command controls the relative size of the quadrilateral elements in the shell mesh.

body <body_id_range> stair block <block_id>

This command sets the id of the SHELL4 element block used to write out the quadrilateral shell mesh.

volume <volume_id_range> scheme stair

Designates a volume to be surrounded by the quadrilateral shell mesh. The volume is meshed using the normal meshing command.

Figure 5-34 shows an example where the Thunderbird is surrounded by an axis-aligned,

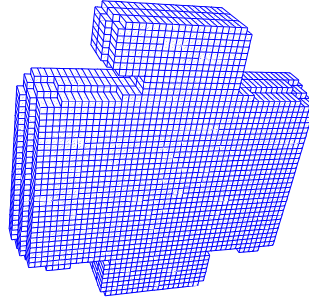


Figure 5-33 StairTool mesh.

structured, non-intersecting shell of quadrilaterals.

Whisker weaving

Whisker weaving is based on information contained in the Spatial Twist Continuum (STC), which is the geometric dual of an all-hexahedral mesh. Whisker weaving begins with a three dimensional geometry and an all-quadrilateral surface mesh, then constructs hexahedral element connectivity advancing from the boundary inward. After the generation of hex mesh connectivity in dual space, this connectivity is then converted into an actual mesh and smoothed to fit the volume. For more details about the whisker weaving algorithm, see [Ref].

The whisker weaving algorithm is under research and development, and its capabilities are limited at the time of writing. Examples of meshes generated using the whisker weaving algorithm are shown in Figure 5-34.

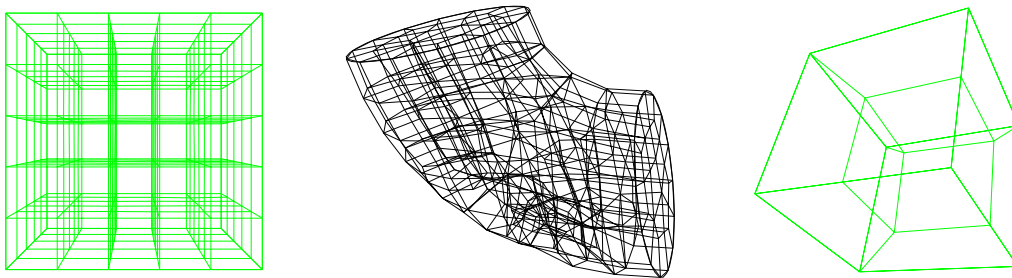


Figure 5-34 Whisker weaving meshes.

To mesh a volume with whisker weaving, the following steps must be taken:

- Set the meshing scheme for the volume to weave.
- Set the smoothing scheme for the volume to Laplacian.
- “Weave” the volume, generating a collection of interdependent “sheets” that are dual to the hex-mesh connectivity.
- Make local connectivity modifications, to improve mesh quality.

- Convert connectivity information to an actual mesh.

Whisker weaving basic commands

This section assumes that there is already a standard volume defined by a geometric model in CUBIT. There is also the option of importing a surface mesh to define the geometry; see “Example:” on page 120.

The basic commands for meshing a volume using whisker weaving are the following. The volume is woven by setting the scheme and issuing the command to mesh the volume:

Volume <volume_id_range> Scheme Weave

Mesh volume <volume_id_range>

Before generating the primal, that is before constructing hexes, the user has the chance to observe, experiment, and resolve knives and other degeneracies in the weave (these options will be described in the next few sections). After inspecting the weave, and assuming there are no degeneracies besides knives and doublets in the weave¹, the primal is generated by setting the volume smooth scheme and issuing the primal command:

Volume <volume_id_range> Smooth Scheme Laplacian

Primal Volume <volume_id>

The resulting mesh can be inspected, modified, and written to a Genesis file using all the typical operations described elsewhere in this manual.

Viewing the Weave

The connectivity information generated by whisker weaving is represented as a series of interdependent, 2D sheets. To view the collection of sheets for a volume, the following command is used:

Draw Arrangement <volume_id>

A particular sheet may be drawn using the command

Draw Sheet <sheet_id>

An example sheet diagram is shown in Figure 5-35 left. The id number for the sheet is drawn in the upper right corner of the diagram. The outer boundary of the sheet represents a “loop” or cycle of mesh quadrilaterals on the geometric boundary² as in Figure 5-35 right. The loop is intersected by chords, each labeled with the face id (outside the loop) and the other sheet number (inside the loop). Hexahedra on the chords are labeled with the hex id number. Self-intersecting chords and knife chords are drawn in a different color from the other chords; the exact color depends on the volume’s mesh color. The user can highlight a chord on a particular sheet with the command

Draw Chord <face_id>

This command highlights the chord on its first sheet; preceding the <face_id> with a minus sign highlights the chord on its second sheet.

-
1. Degeneracies (knives, doublets, etc) are generated as a natural part of whisker weaving, and some are left in the mesh for resolution after weaving is complete. Some of these degeneracies must be resolved before a primal can be constructed; see “Resolving whisker weaving degeneracies” on page 115.
 2. Some sheets have more than one loop; additional loops are shown as inner boundaries on the sheet diagram.

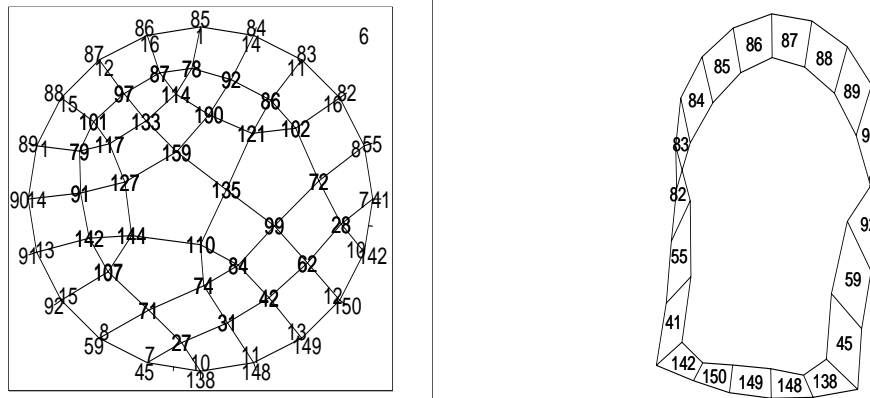


Figure 5-35 Example sheet diagram (left) and corresponding loop (right).

The quadrilaterals forming the loop(s) of a sheet are drawn by the following command; see Figure 5-35 right.

Draw Loop <sheet_id>

Groups of STC entities can be displayed in separate graphics windows. The window positions may default to on top of one another, so the user may have to move a window before realizing that there are windows underneath it. To enable this option, use the command

Set Windows On

The user can obtain more information about various STC entities by using the listing commands. The following commands list information about sheets, chords, sheetchords, hexes, and doublets, respectively:

List Sheet <sheet_id>

List Chord <chord_beginning_face_id>

List Schord <chord_beginning_face_id>

List Whex <hex_id>

List Doublets

For information about the output of these commands, contact the CUBIT development team.

Resolving whisker weaving degeneracies

Sometimes the STC or weave contains invalid connectivity, also referred to as *degeneracies*. Care is taken to resolve these degeneracies automatically, but sometimes there is a choice for the user, and in some cases the automatic resolution has not yet been implemented. The options in this section are used to resolve a number of types of degeneracies.

• Knife resolution

A knife element is a degenerate hexahedron with one face collapsed by merging a pair of opposite nodes. In the weave, a knife appears as the middle hex of a chord whose beginning and ending face are the same. In actuality, this chord passes through the same face on both ends, passes through the same hexes in the same order, and then terminates inside the volume at the knife element.

Knives may be resolved by pushing them or pulling them through the mesh [Ref], or knives may be left in place and a hex-plus-knives mesh can be generated. Pulling knives, or collapsing them, is the most robust, but always changes the surface mesh and sometimes generates other degeneracies. Knife driving in the normal manner is not always possible. An alternative algorithm for driving knives, called “superdrive”, is also being investigated. Superdriving results in STC invalidities whose automatic resolution is not currently implemented.

If the user wants a mesh without any knives, the simplest thing to do is collapse all the knives before primal construction, using the command:

Whiskerknife Collapse all

Better mesh quality may be obtained by driving knives together or to the surface mesh instead, using the command

Whiskerknife Drive all

Knives are first collapsed to just inside the surface, in order to maximize the chance of being able to drive them in the normal fashion. If its impossible to drive a knife, then it is collapsed.

The following command may be used to collapse or drive knives for entire sheets:

Whiskerknife {Drive|Collapse} [Sheet] [<sheet_id_range> | All] [One]

If none of the options are specified, the first knife found is resolved (by driving or collapsing). If individual sheets are specified, then all, or if the keyword “one” is specified then just one, of the knives on those sheets are resolved. If the range is “all”, then all sheets have all knives resolved. Note that collapsing a knife results in the creation of a new sheet, and other knives on the original sheet may end up on the new sheet.

The following commands may be used to collapse knives individually:

Whiskerknife Collapse Chord <face_id> [Number <n>]

This collapses the knife whose base chord begins at the specified face. If the number option is used, the knife is collapsed <n> hexes.

To drive a knife, the following command is used:

Whiskerknife Drive Chord <face_id> [Report]

This causes the specified knife to be resolved by driving if possible, otherwise by collapsing. If the report option is used, a tree of possible drives for the knife chord is given. The knife is not backed up first, but this can be done manually with the **Whiskerknife Collapse Chord Number <n>** command, with <n> set to exactly the number of hexes on the chord. The possible drive path(s) are reported using paths of STC edges on the sheet diagram. Each edge is also reported to the list output as a pair of vertices, each vertex dual to either a hex or a surface mesh face. The user then chooses one of the locations reported, and tells CUBIT to drive the knife to that location.

Whiskerknife Drive Chord <face_id> [Hex|Face <a1> Hex|Face <a2> Hex|Face <b1> Hex|Face <b2>]

The optional parameters specify a specific quadruple of entities for the knife to drive to, from its current location. It is recommended that the user specify only locations obtained by the **Whiskerknife Drive Chord <face_id> Report** command.

Another option for resolving knives is called “superdrive”. This method converts the knife to a related degenerate element which can be driven more easily through a mesh. This technique produces degenerate elements in the wake, but can be used to drive knives on the same sheet

together. The degenerate elements in the wake can be resolved using pillowing, described in the next section. To allow the use of superdriving with any of the drive commands described above, set the following option:

Set Superdrive {On|Off}

Knives can also be created in a mesh by collapsing a face on the surface of a completed weave. This capability is useful for studying knife behavior in particular situations. To produce a knife by collapsing a surface face, use the following command:

Whiskerknife Collapse Chord <face_id> [Node <node_id>] [Number <n>]

If a node is specified, then the face is collapsed by merging that node with the node at the opposite corner of the face, otherwise the node pair is chosen at random. The base face of the new knife chord will be the other surface face of the chord through the collapsed face. If <n> is specified, that many hexes on the knife chord are collapsed (<n> may be set to zero); otherwise, the chord is completely collapsed.

• Doublet Resolution

Doublets are two faces sharing two edges, or two hexes sharing two faces (which implies the former case). Surface doublets are formed when two faces of a hex share an edge and lie on the same planar or near-planar surface. Doublet resolution sometimes changes the surface mesh. To resolve doublets in a weave, first construct the primal. Doublets that occur in any mesh within CUBIT, including those arising from whisker weaving and plastering, may be resolved with the following command:

Pillow Volume <volume_id>

Pillowing doublets deletes any sheets for the volume. The basic strategy is to use abstract pillow sheets to locally refine the mesh, putting a layer or two of hexes between the shared faces, or between the surface doublet faces and the hex that contains them.

Pillowing doublets may be performed in meshes that contain knives. The user should be aware that knives by definition contain two faces sharing two edges, and since both of these faces lie in the same “hex” (i.e. the knife), pillowing doublets cannot remove such doublets.

• Degeneracy Resolution Using Pillow Sheets

Pillow sheets, which are sheets with no loops lying entirely inside the volume, are useful for resolving a variety of degeneracies produced by whisker weaving. They may also be useful for adaptive mesh refinement. For example, through-cells and through-chords, STC entities that pass completely through the volume from one part of the surface mesh to another, can be resolved with pillow sheets¹.

The placement of pillow sheets to resolve various types of degeneracies is still being researched. In some cases, pillow sheets can be placed automatically; in other cases, the pillow sheets must be allocated, then placed to separate specific entities in the STC, and then inserted and verified manually by the user.

1. Through-chords are chords which extend across the sheet without intersecting any other chords. Through-2-cells are 2-cells containing 2 or more distinct edges on the loop. Through-3-cells, which cannot be observed directly on a single sheet diagram, are 3D analogies of through-2-cells. See [Ref] for more details.

To automatically insert a pillow sheet that hugs the surfaces of the specified volume, in effect inserting a buffer layer of hexes and cutting all cells that touch the surface, including though-cells, the following command is used:

Pillow Weave <volume_id_range> Buffer

An example of a non-distinct STC entity invalidity is a 2-cell that contains the same STC edge twice. This corresponds to a mesh edge being contained twice by a mesh quadrilateral. These types of invalidities are detected and resolved automatically with pillow sheets by the following command.

Pillow Weave <volume_id_range> Automatic

The user can insert pillow sheets interactively to resolve degeneracies and refine the mesh before primal construction. A sequence of commands is used to allocate the pillow sheet, place it, then insert it in the weave. The commands (in sequence) are

Pillow Weave <volume_id> Create

This creates a vacuumous pillow sheet. The id of the new pillow sheet is reported.

A pillow sheet is placed next to specified faces, and surrounding specified hexes, using the following commands:

Pillow Sheet <sheet_id> Face <face_id1> [<face_id2>...]

Pillow Sheet <sheet_id> Hex <hex_id1> [<hex_id2>...]

After specifying the placement of the pillow sheet, the sheet is inserted in the weave, cutting around the prespecified hexes and cutting next to the prespecified faces, using the following command:

Pillow Sheet <sheet_id> Cut

The following commands perform datastructure cleanup and correctness verification for the pillow sheet:

Pillow Sheet <sheet_id> Orient

Pillow Sheet <sheet_id> Verify

Weaving without geometry

Typically, whisker weaving is used to mesh a solid model which resides in the CUBIT database. However, whisker weaving can also work from a quadrilateral mesh defining the boundary of the volume to be meshed. This surface mesh can be read directly into CUBIT using the following command:

Import Geometry Mesh '<exodusII_filename>' Block <block_number>

This command creates one volume and one surface, regardless of any geometric “corners” or “edges”. No curves or vertices are created. The id’s of the volume and surface are reported to the list output. Currently there is no body containing the volume. In addition, a large brick containing the extremes of the surface mesh is created, and assigned to a new volume. This brick is used for graphics auto scaling.

The resulting volume may be looped and meshed with whisker weaving. Meshing the volume using whisker weaving works as before, with the exception that whiskerknife collapsing doesn’t

work well (because it involves changing the surface mesh). Smoothing is also limited for this reason.

Miscellaneous Whisker Weaving Options

Most of these options are of interest only to developers, but a sophisticated user may want to experiment with some of them.

• Whisker Weaving Sub-Command Interface

Whisker weaving can be viewed and controlled in a fine-grained fashion using the whisker weaving sub-command interface. This interface is of interest mainly to developers, but it can be useful for observing and debugging whisker weaving at a low level. To enable the sub-command interface, execute the following command before weaving begins:

Set Query On

This causes whisker weaving to pause after every whiskerhex is woven, and draw the three sheets passing through that hex. The user can then enter a series of commands for inspecting the STC information or for controlling the progress of weaving. The commands available at the sub-command line interface are listed in Table 5-1, and are listed by pressing ‘h’ for help. The sub-command interface can be disabled by entering the ‘t’ command.

Table 5-1. Whisker weaving sub-command line interface commands.

Syntax	Command
2 <int>	construct 2-cells for sheet <int>
a	activate mouse panning/zooming mode
b	break to command line
c	continue hexing
d <int>	draw sheet <int> (0 for all sheets)
e	drive all wedges (after weaving)
f <int>	drive one wedge on sheet <int> (0 for first encountered)
g <f> <v>	set debug flag f to value v
j	skip chord joins for next hex
l <m> <n>	locate chord with face m, n'th sheet
m <int>	smooth sheet <int>
o <int> <double>	outwardly expand sheet <int> drawing by <double> [> 0.0]
q <3 ints>	quadrant draw 3 sheets
s <int>	sleep for <int> hexes
t	toggle sheet printing and continue

Table 5-1. Whisker weaving sub-command line interface commands.

Syntax	Command
v <int>	validate sheet <int> (0 for all sheets)
z	dump postscript file to ‘out.ps’

• **Explicit Looping**

Creating the loops, which are the STC of the surface mesh, is the first step of weaving. Normally the loops are created at the start of whisker weaving; however, the user may explicitly create loops if desired. This may be done regardless of whether the volume’s meshing scheme is weaving, so the user can examine the STC before deciding on the meshing scheme. The following command is used to loop the volume:

Loop Volume <volume_id>

Each loop is created by traversing from one edge of a quadrilateral to the opposite side, until the first edge is reached again. The sheets and loops may be viewed with the basic commands described in “Viewing the Weave” on page 114. Looping, and hence weaving, currently only work with manifold geometry.

• **Deleting loop self-intersections**

A loop self-intersection is a quadrilateral that a single loop passes through twice, once through each pair of opposite edges. Such quadrilaterals are very common in irregular surface meshes. Self-intersections have a number of consequences, for example knives only arise on chords that start at self-intersections. If looping has already been done, self-intersections can be removed with the command:

Delete Intersections <volume_id_range>

Alternatively, setting a flag will run a surface-mesh modification algorithm as the last step of looping a volume. This flag is set using the command:

Set Uncross On

The algorithm used to remove self-intersections collapses certain surface faces, so that no loop of the surface STC self-intersects. This algorithm employs a heuristic so that in practice only about the square root of all self-intersecting quadrilaterals need to be collapsed. The heuristic also tries to make the surface mesh connectivity as regular as possible.

Deleting self-intersections removes some of the complexity of the surface mesh, but it’s not clear that this benefits the whisker weaving algorithm. This will be an area of further research.

Example:

```
brick width 10
volume 1 interval 3
mesh surface 1 to 6
block 1 surface 1 to 6
export genesis “block3-3-3.gen”
display
```

```

pause
reset
import geometry mesh "block3-3-3.gen" block 1
display
volume 1 scheme weave
set primal on
set query off
volume 1 smooth scheme laplacian
mesh volume 1
display

```

• **Weaving database while plastering**

The whisker weaving database may provide insight when running unstructured hexahedral meshing algorithms other than whisker weaving, in particular plastering. To enable the creation of whisker weaving data during the operation of plastering, the following command is used:

Set Weavedata On

STC entities can be drawn after each hex is generated by plastering; to enable this option, the following command is used:

Set Step On

To draw either the arrangement of all of the STC sheets and/or the three sheets passing through the hex just created, the following commands are used, respectively:

Set Arrangement {On|Off}

Set Sheets {On|Off}

Note that the sheets and the arrangement can be drawn in separate windows, using the **Set Windows On** command described earlier.

Dicing

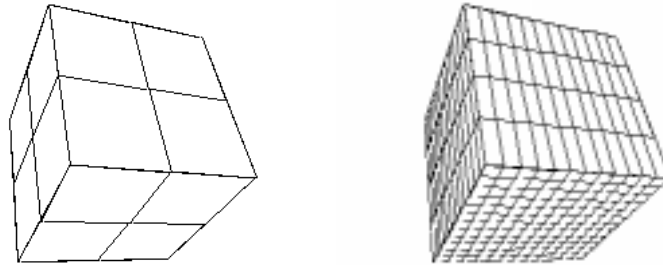
There are situations in which it is advantageous to generate the desired finite element mesh by first generating a coarse mesh and then refining that mesh. This is true for several reasons:

- It is often easier to control details of the structure of a coarse mesh than a fine one.
- Coarse meshes take less time to generate. This allows modifications to be made more quickly.
- Some mesh generation algorithms require large amounts of memory for each element in the mesh. This can lead to limits on the resolution that can be attained with a given algorithm.

The dicing algorithm maintains the overall structure defined by a coarse mesh while increasing the resolution of the mesh. Refinement is done in such a way as to generate additional elements quickly while utilizing computer memory efficiently. Dicing can be applied to volume meshes and surfaces meshes. It is expected that dicing will be most commonly used to refine volume meshes, so it is presented here. The same principles and commands also apply to surface dicing.

The volume mapping algorithm is best-suited for hexahedral regions. Therefore, each element in an all-hexahedral mesh is a mappable subregion of the volume in which it is located. Dicing refines coarse meshes by treating each element in the coarse mesh as a mappable sub-region.

Each coarse element is replaced by a structured grid of fine elements generated by a mapping algorithm. A simple example is shown in the following figures:



• *Dicer Sheets and Refinement Intervals*

The number of elements that will replace each coarse element is determined by the element's refinement intervals. A *refinement interval* is the number of fine mesh edges that will replace a given coarse mesh edge. A different refinement interval can be set for edges along each axis in the coarse element's natural coordinate system. In the example above, the volume has a refinement interval of two in one direction and six in the other two directions.

To generate a conformal mesh, groups of coarse mesh edges must have the same refinement interval. Specifically, mesh edges on opposite sides of a mesh face must have the same refinement interval. Because mesh edges can be a part of multiple mesh faces, refinement interval dependencies propagate through the mesh. Groups of coarse edges that must have the same refinement interval are called *Dicer Sheets*. Coarse mesh edges are automatically grouped into Dicer Sheets when necessary.

• *Dicing Basic Commands*

Before Dicing can be applied to a mesh, a coarse mesh must first be generated using one of the algorithms described in the chapter on mesh generation. Once a coarse mesh exists, refining that mesh with Dicing is very similar to generating a mesh with other algorithms. The scheme for the entity whose mesh will be refined is first set to *Dice*:

{volume | surface} <id_range> Scheme Dice

The refinement interval for the entity is then set:

{volume | surface | curve} <id_range> Interval <refinement interval>

When an interval is set for a given entity, each DicerSheet with at least one edge in or on that entity has its refinement interval set to the specified number. The last command affecting a given DicerSheet determines that DicerSheet's refinement interval. As an example, assume that the refinement interval for a volume is set to 3. Setting the refinement interval for one of that volume's curves to 5 would change the refinement interval for all DicerSheets passing through the specified curve. However, if the same operations were performed in the opposite order (first set the interval on the curve to 5, then set the interval on the volume to 3), ALL Dicer Sheets would have a refinement interval of 3.

If no refinement interval is specified, the default refinement interval is used. The default interval is initially set to two, but it can be set to any integer value. The default refinement interval is set with the following command:

DicerSheet Default Interval <interval>

After refinement intervals have been set, a fine mesh can be generated with the following command:

Mesh {volume | surface} <id_range>

At this point, only the fine nodes have been generated, not the fine elements. In order to see the fine mesh, node visibility must be turned on:

Node Visibility On

After viewing the fine nodes, the coarse mesh may be replaced with the fine mesh:

Replace Mesh {volume | surface} <id_range>

• Additional Dicing Commands

Several utilities have been developed to assist the user during the refinement process.

Coarse mesh edges may be grouped into Dicer Sheets, even if the scheme for that entity is not *Dice*, using the following command:

{volume | surface} <id_range> Initialize Dicer

It is often useful to determine which DicerSheet a coarse mesh edge belongs to, and which other edges belong to the same DicerSheet. With the following command, the mouse may be used to click on a coarse edge in the graphics window. The number of the DicerSheet the edge belongs to will be printed, and all edges in the same DicerSheet will be highlighted:

Pick DicerSheet [multiple]

All the edges in the specified Dicer Sheet may be highlighted with the following command:

Highlight DicerSheet <id_range>

The refinement interval for a specific DicerSheet can be set individually with the following command:

DicerSheet <id_range> Interval <interval>

The following command deletes the fine nodes generated by the Dicer. This command only works before using the **Replace Mesh** command. Fine nodes that rely on the deleted fine nodes are also deleted. For example, if the fine nodes on a surface are deleted, the fine mesh on any attached volumes is invalid. The fine nodes in those volume are therefore deleted along with the nodes on the surface. If the optional **Propagate** keyword is used, the fine mesh will be deleted from any child entities as well.

Delete Fine Mesh {volume | surface | edge} <id_range> [Propagate]

To list the number of DicerSheets in the model, along with their ID numbers, type:

List DicerSheet

To list information about a specific DicerSheet, type:

List DicerSheet <id>

Generating the Volume Mesh

Once the desired scheme has been chosen, the volume can be meshed. If the user wishes to receive a different element type than the default (eight-node hexahedrons) for volume meshing, this specification needs to be set prior to creating any volume meshes. The command to mesh a volume is:

mesh volume <range> | All

A body can contain a number of volumes, although it normally only contains one. The following command will mesh each volume owned by the body (bodies) specified:

mesh body <range> | All

The resulting mesh will be drawn on the screen in the mesh color designated for the volume which was meshed.

▼ Mesh Editing

Mesh editing capability exists in the areas of mesh smoothing, mesh deletion, and movement of specific nodes and nodesets. A limited capability to modify portions of a mesh is provided to allow the user to make judgements about the level of smoothing required for volumetric meshes. Since the models to be meshed vary widely, and since CUBIT does not contain a sophisticated geometry recognizer, CUBIT is unable to make decisions regarding the type of smoothing algorithm to employ. Certain algorithms work well for classes of problems and fail for others. There seems to be no perfect smoothing algorithm, therefore the decision of which type of smoother to use is left to the user.

Mesh Smoothing

Surface smoothing

Surface smoothing algorithms currently consist of a variety of equipotential stencils, length-weighted laplacian, and centroid area pull. The nature of equipotential smoothers is one of weight equalization between adjacent nodes. For a generic, area, or Jacobian-based weighted smooth, this is roughly similar to equalizing areas between adjacent elements. The techniques behave well for regular or irregular grids on non-periodic surfaces, but are not yet released for periodic surfaces such as cylinders, spheres, (some) nurbs, and tori.

The Laplacian smoothing approach calculates an average element edge length around the mesh node being smoothed to weight the magnitude of the allowed node movement [8]. Therefore this smoother is highly sensitive to element edge lengths and tends to average these lengths to form better shaped elements. However, similar to the mapping transformations, the length-weighted Laplacian formulation has difficulty with highly concave regions⁸.

The Centroid Area Pull smoothing approach attempts to create elements of equal area by. Each node is pulled toward the centroids of adjacent elements by forces proportional to the respective element areas [8].

Smoothing is implemented like the meshing, where the scheme is set first and the action performed later, with separate commands. The command line syntax for setting the smoothing scheme for a surface is as follows:

**Surface <range> Smooth Scheme Equipotential [Fixed]
[Weight {Jacobian | Area | Inverse [Area]}]**

Surface <range> Smooth Scheme Laplacian [Fixed]

Surface <range> Smooth Scheme Centroid Area Pull[Fixed]

If the **Weight** keyword is not specified, a **Generic** weighting is used by default. The **Fixed** keyword forces the nodes lying on the bounding curves of a surface to remain stationary instead of “floating” along the equation of the curve until all nodes have converged. Note that this restriction limits the amount of impact the smoothing operation can have on the surface mesh.

To smooth a surface based on the previously set scheme, the following command is used:

Smooth Surface <range> [global]

If no scheme has been set, the Equipotential scheme is used by default. The optional global identifier is only valid with the laplacian and centroid area pull smoothing schemes. If entered, all surfaces specified by range will be smoothed at one time. If global is not specified, the surface will be smoothed sequentially.

Volume smoothing

Two volume smoothing algorithms are currently available in CUBIT. The first type of user controlled smoother is the length-weighted Laplacian. This smoothing approach calculates an average element edge length around the mesh node being smoothed to weight the magnitude of the allowed node movement [8]. Therefore this smoother is highly sensitive to element edge lengths and tends to average these lengths to form better shaped elements. However, similar to the mapping transformations, the length-weighted Laplacian formulation has difficulty with highly concave regions⁸.

The second type of smoother is a variation of the equipotential [8] algorithm that has been extended to manage non-regular grids [9]. This method tends to equalize element volumes as it adjusts nodal locations. The advantage of the equipotential method is its tendency to “pull in” badly shaped meshes. This capability is not without cost: the equipotential method may take longer to converge or may be divergent. To impose an equipotential smooth on a volume, each element must be smoothed in every iteration—a typically expensive computation. While a Laplacian method can complete smoothing operations with only local nodal calculations, the equipotential method requires complete domain information to operate.

Smoothing is implemented like the meshing, where the scheme is set first and the action performed later, with separate commands. The command line syntax for setting the smoothing scheme for a volume is as follows:

**Volume <range> Smooth Scheme
{Laplacian | Equipotential} [Fixed]**

The **Fixed** keyword force the nodes lying on the bounding surfaces of a volume to remain stationary instead of “floating” along the equation of the surface until all nodes have converged. Note that this restriction limits the amount of impact the smoothing operation can have on the volume mesh.

To smooth a volume based on the previously set scheme, the following command is used:

Smooth Volume <range>

If no scheme has been set, the **Equipotential** scheme is used by default.

While future objectives include investigation into weighting schemes to explicitly control mesh flow according to user-defined field or element functions, a simple hex weighting function exists which demonstrates the potential of the equipotential smoothers. This command applies a user specified weight to a group of hex elements, in this case, the elements which contain a mesh face which belongs to a specified geometric surface. By adjusting the weight and running the smoother, one can expand or compact the elements being weighted. This capability may be used eventually to control adaptive hex element meshing, and to assist the free-form three-dimensional volumetric algorithms in their efforts to maintain element quality. The command syntax for applying the hex element weights is as follows:

Weight Hexes Surface <range> <weight>

Mesh Deletion

Complete mesh removal

The command to remove all existing mesh entities from the model is:

delete mesh

Partial mesh removal

Additional commands are available for deleting only selected portions of the mesh in the current CUBIT model. Partial delete capabilities exist for volumes, surfaces, curves, and vertices. Only the mesh which is owned by or is dependent on the specified geometry will be removed.

The routines are intelligent in the respect that if a mesh delete command is executed for a surface, curve, or vertex which belongs to one (or more) fully meshed volumes, CUBIT will delete all mesh entities which must be deleted as a result: for example, if one node on the vertex is deleted, then the mesh which lies on the connected curves to that vertex is incomplete and will be deleted, and then the surface mesh which relied on the curves must also be deleted, and finally the interior hex elements within the volume. The command syntax for these commands is as follows:

delete mesh volume <range> [Propagate]

delete mesh surface <range> [Propagate]

delete mesh curve <range> [Propagate]

delete mesh vertex <range> [Propagate]

These commands automatically cause deletion of mesh on higher dimensional entities owning the target geometry, but they do not cause a deletion of the mesh on lower dimensional ones. For example, deletion of mesh on a surface would not affect the mesh on the curves and vertices bounding that surface. To force mesh deletion on all such lower dimensional geometries, add the command “propagate” at the end of the command line. This forces the delete mesh command to propagate downward to all lower level entities until the vertices are reached. An exception to this chain of propagation occurs if a geometry is reached that is owned by another higher level geometry which is still meshed. This exception prevents inadvertent destruction of neighboring meshes.

Individual mesh face removal

Mesh faces can be deleted individually using the **Delete Face** command. This command closes a face by merging two mesh nodes indicated in the input. The syntax for this command is:

Delete Face <face_id> Node1 <node1_id> Node2 <node2_id>

This command is provided primarily for developers' use, but also provides the user fine control over surface meshes. At the present time, this command works only with faces appearing on geometric surfaces and should be used before any hex meshing is performed on any volume containing the face to be deleted.

Node and NodeSet Repositioning

A capability to reposition nodesets and individual nodes is provided. This capability will retain all the current connectivity of the nodes involved, but it cannot guarantee that the new locations of the moved nodes do not form intersections with previously existing mesh or geometry. This capability is provided to allow the user maximum control over the mesh model being constructed, and by giving this control the user can possibly create mesh that is self-intersecting. The user should be careful that the nodes being relocated will not form such intersections.

The user can reposition nodes appearing in the same nodeset using the **NodeSet Move** command. Moves can be specified using either a relative displacement or an absolute position. The command to reposition nodes in a nodeset is:

nodeset <id> move <delta_x> <delta_y> <delta_z>

nodeset <id> move to <x_position> <y_position> <z_position>

The first form of the command specifies a relative movement of the nodes by the specified distances and the second form of the command specifies absolute movement to the specified position.

Individual nodes can be repositioned using the Node Move command. Moves are specified as relative displacements. The command syntax is:

Node <range> Move <delta_x> <delta_y> <delta_z>

▼ Mesh Importing and Duplicating

Limited capability exists to instantiate new mesh by importing mesh from an external file and by copying mesh from inside the current mesh model.

Importing mesh from an external file

A limited capability exists to read in a previously created mesh from an existing ExodusII file and associate the mesh (retaining full functionality for additional meshing or smoothing operations which depend on the imported mesh) with matching geometry in the CUBIT model. This command is useful for continuing a previous meshing problem at the point it was terminated rather than regenerating the entire mesh from the beginning. The geometry which matches the original mesh must be present in the model for the command to work.

Two methods have been implemented for associating a mesh with a geometry. The first method is proximity-based, that is it compares node positions with those of vertices, curves, etc., to

determine the nodal associativity. This method has been known to fail when computing ownership on periodic curves and surfaces, and may not be robust in cases where the proximity test is not sufficiently restrictive. This method has been retained mainly for compatibility with old meshes.

The second associativity method is based on logical associativity data stored in the ExodusII file. Each geometry entity has a corresponding nodeset which contains the nodes owned by that entity, and nodesets are associated back to geometry entities using geometry entity names (see “Entity Names” on page 79). (To store associativity information in the ExodusII file, the **NodeSet Associativity** command must be entered before issuing the **Export Genesis** command; see “Nodeset Associativity Data” on page 136.) Note that before importing a mesh to be associated with a geometry, the geometry must be in the same state it was in when the mesh block was written. That is, the topology must be the same, and there must be corresponding named geometry entities for each of the entities owning mesh in the original problem. This can be done either by executing the same geometry generation sequence, or by arriving at the geometry using a different method and explicitly naming the geometry entities to match those in the original problem. Only the names are used to find matching geometry entities (i.e. no check is made to ensure that the entity types match).

The same command line syntax is used for both associativity methods. The second (logical) method is attempted first; if the logical associativity information is not located in the ExodusII file, the second method is used. The command line syntax for importing a mesh into CUBIT is:

```
Import Mesh 'exodusII_filename' Block <block_id> Volume <volume_id>
```

Duplicating mesh

If the geometry to be meshed was generated using the body copy command explained in “Copy Bodies” on page 52, then the mesh from the original geometry can be copied directly to the new geometry using the command

```
copy mesh {volume|surface} <id1> onto {volume|surface} <id2>
```

Note that if the copied bodies have had any features merged (co-incident surfaces or curves joined), then the new bodies will no longer be merged, and the co-incident surfaces and curves will exist once again. The **Copy Mesh** command similarly will not maintain the merged feature structure, and mesh will appear on both co-incident surfaces if multiple bodies were copied which share surfaces. The solution is to perform merging on bodies after they have copied, and to only use **Copy Mesh** for volumes and surfaces which do not share any co-incident features with adjacent geometry.

▼ Mesh Quality

The ‘quality’ of a mesh can be assessed using the element quality functions. These functions calculate several element shape factors which may have an affect on the accuracy of the finite element results calculated using the element.

Background

The quadrilateral element quality metrics that are calculated are aspect ratio, skew, taper, warpage, element area, and stretch. The calculations are based on an article by John Robinson entitled “CRE method of element testing and Jacobian shape parameters,” Eng. Comput., 1987,

Vol. 4. An illustration of the shape parameters is shown in Figure 5-36 The warpage is calculated

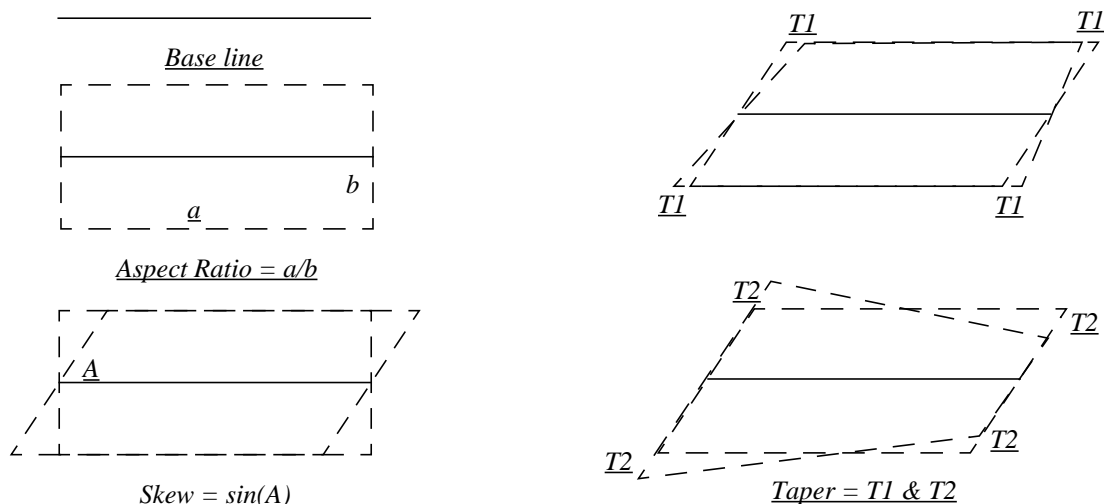


Figure 5-36 Illustration of Quadrilateral Shape Parameters (Quality Metrics)

as the Z deviation from the 'best-fit' plane containing the element divided by the minimum of 'a' or 'b' from Figure 5-36. The stretch metric is calculated by dividing the length of the shortest element edge divided by the length of the longest element diagonal.

The hexahedral element quality metrics that are calculated are aspect ratio, skew, taper, element volume, stretch, and diagonal ratio. The calculation of these metrics is similar to that used for the quadrilateral elements. A good illustration and discussion of the mode shapes for an eight-node hexahedral element can be found in Chapter 3 of L. M. Taylor and D. P. Flanagan, "PRONTO 3D: A Three-Dimensional Transient Solid Dynamics Program," SAND87-1912.

Command Syntax

The commands to access the quality metrics are:

quality <entity_list> [global]

quality <entity_list> [global] display|draw 'metric_name'

The **global** identifier indicates that all specified entities are to be treated as a single entity instead of as several distinct entities. Valid values for the **metric_name** identifier are: **Aspect Ratio**, **Skew**, **Taper**, **Element Area** (Quad Only), **Element Volume** (Hex Only), **Warpage** (Quad Only), **Stretch**, and **Diagonal Ratio** (Hex Only). The example section below shows the typical output.

Command Examples

quality surf all global

-- lists quality summary for all surfaces in model. One summary

quality surf all

-- lists quality summary for all surfaces in model. One summary per entity

quality group 1

-- lists quality for the RefEntities in the group. Determines the highest common dimension (hex/quad).

quality surf 1 surf 2 surf 9

-- lists summary for surfaces 1, 2, and 9

quality surf all global draw 'Aspect Ratio'

-- Draws color-coded plot of the aspect ratios of the element faces.

Example Output

The typical summary output from the command **quality surface 1** is shown in Table 5-1. Figure 5-37a shows the histogram output corresponding to the above summary. The colored

Table 5-1 Sample Output for 'Quality' Command

Surface 1 Element Metrics:					
Function Name	Average	Std Dev	Minimum (id)	Maximum (id)	
Aspect Ratio	1.272e+00	2.336e-01	1.000e+00 (86)	2.200e+00 (433)	
Skew	2.035e-01	1.790e-01	7.168e-04 (121)	7.778e-01 (280)	
Taper	1.529e-01	1.048e-01	4.783e-03 (254)	6.842e-01 (70)	
Warpage	0.000e+00	0.000e+00	0.000e+00 (1)	0.000e+00 (1)	
Element Area	5.244e-04	5.683e-04	3.305e-05 (154)	2.371e-03 (229)	
Stretch	7.467e-01	1.136e-01	2.983e-01 (433)	9.648e-01 (310)	

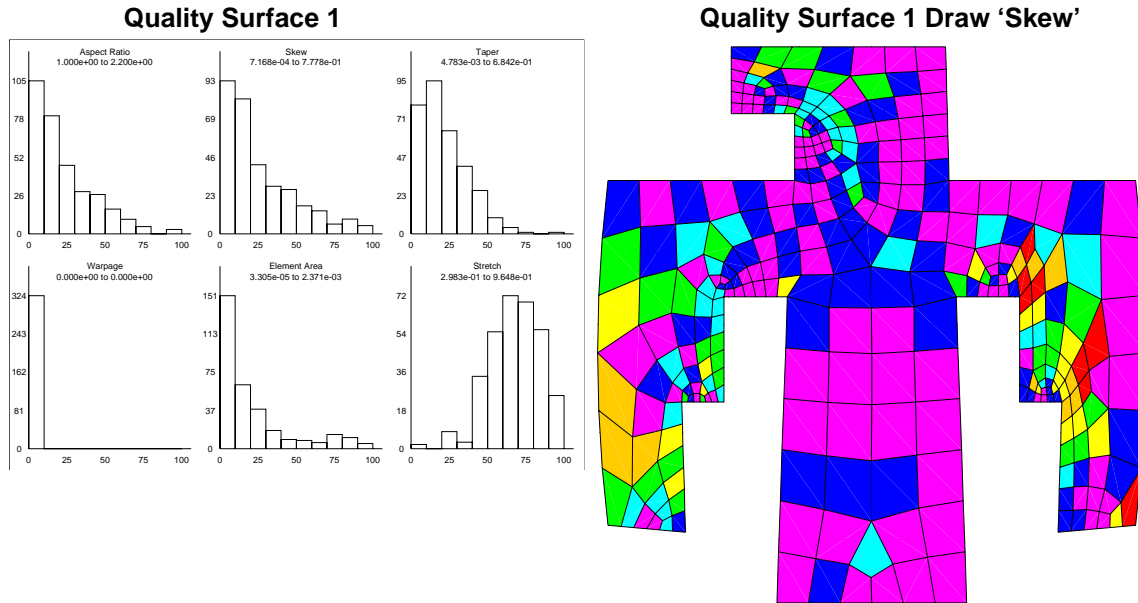
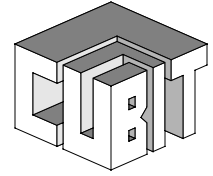


Figure 5-37 Illustration of Quality Metric Graphical Output

element display resulting from the command **quality surface 1 draw 'Skew'** is shown in Figure 5-37b. In addition, a legend () is output to the terminal.

Table 5-1Element Quality Plot Legend

Magenta ranges from 7.168e-04 to 1.117e-01 (127 entities)
Blue ranges from 1.117e-01 to 2.227e-01 (82 entities)
Cyan ranges from 2.227e-01 to 3.338e-01 (41 entities)
Green ranges from 3.338e-01 to 4.448e-01 (35 entities)
Yellow ranges from 4.448e-01 to 5.558e-01 (20 entities)
Orange ranges from 5.558e-01 to 6.668e-01 (11 entities)
Red ranges from 6.668e-01 to 7.778e-01 (8 entities)



Chapter 6: OvFinite Element Model Definition and Output

- ▼ Finite Element Model Definition...133
 - ▼ Element Block Specification...134
- ▼ Boundary Conditions: Nodesets and Sidesets...136
 - ▼ Setting the Title...137
- ▼ Exporting the Finite Element Model...137

This chapter describes the techniques used to complete the definition of the finite element model and the commands to export the finite element mesh to an Exodus database file. The definitions of the basic items in an Exodus database are briefly presented, followed by a description of the commands a user would typically enter to produce a customized finite element problem description.

▼ Finite Element Model Definition

Sandia's finite element analysis codes have been written to transfer mesh definition data in the ExodusII [6] file format. CUBIT is one code in a suite of computer codes that supports the ExodusII format for the preprocessing and postprocessing of finite element analyses [14]. The ExodusII database exported during a CUBIT session is sometimes referred to as a Genesis database file which is the term used to refer to a subset of an Exodus file containing the problem definition only, i.e., no analysis results are included in the database.

A Genesis database consists of the following basic entity types: Element Blocks, Nodesets, Sidesets, and Property Names..

Element Blocks

Element Blocks (also referred to as simply, *Blocks*) are a logical grouping of *elements* all having the same basic geometry and number of nodes. All elements within an Element Block are required to have the same element type. Access to an Element Block is accomplished through the use of a single integer ID known as the Block ID. Typically, Element Blocks are used by analysis codes to associate material properties and/or body forces with a group of elements.

Nodesets

Nodesets are a logical grouping of *nodes* also accessed through a single ID known as the Nodeset ID. Nodesets provide a means to reference a group of nodes with a single ID. They are

typically used to specify load or boundary conditions on the CUBIT model or to identify a group of nodes for a special output request in the finite element analysis code.

Sidesets

Sidesets are another mechanism by which constraints may be applied to the model. Sidesets represent a grouping of *element sides* and are also referenced using an integer Sideset ID. They are typically used in situations where a constraint must be associated with element sides to satisfactorily represent the physics (for example, a contact surface or a pressure.).

Property Names

▼ Element Block Specification

Element blocks are the method CUBIT uses to group related sets of elements into a single entity. Each element in an element block must have the same dimensionality, type, number of nodes, and number of attributes. Element Blocks may be defined for volumes, surfaces, and curves. Multiple volumes, surfaces, and curves can be contained in a single element block, but a volume, surface, or curve can only be in one element block. Element blocks are defined with the following Block commands.

Block <block_id> {Curve | Surface | Volume} <range>

Block <block_range> Element Type <type>

Block <block_range> Attribute <value>

The **block_id** and the geometry type (curve, surface, or volume) which will be a member of this block

The number following the element name denotes the number of nodes in the element. For example, the **Hex27** element is a 27-noded hexahedral element with mid-side, mid-face, and mid-volume nodes. The Shell and Bar elements require the specification of an **Attribute**¹ value which defines the thickness or cross-sectional area of the element for use in the finite element code². The attribute defaults to 1.0 if not specified. The commands to perform these functions using the command line are:

Where the first command defines a **block_id** containing the specified geometric entities, the second command sets the **Element Type** for that block and the third command sets the **Attribute** for those elements.



Note: Higher order element blocks **must** be specified prior to meshing since additional nodes are inserted as part of the meshing process *only* if an Element Block's element type calls for them.

1. Only zero or one attributes can be defined at the current time. This limitation will be removed in a future version.
2. The thickness and cross-section attribute values are not used internally in CUBIT, they are merely flags which are written to the EXODUS file to be used by subsequent codes. The documentation for the code which will be reading the EXODUS file should be consulted to determine the correct specification and use of the attribute value for the Shell and Bar elements.

Default Element Types, Block IDs, and Attributes

The following defaults will be used unless otherwise specified or modified:

Volume: The default block ID will be set to the Volume ID and 8-node hexahedral elements will be generated.

Surface: The block ID will be set to 0 and 4-node shell elements will be generated.

Curve: The block ID will be set to 0 and 2-node bar elements will be generated.

Meshing could then be accomplished and the desired finite element model exported to the Genesis database.

Element Block Definition Examples

Multiple Element Blocks

Multiple element blocks can and almost always are combined when generating a finite element mesh. For example if the finite element model consists of a block which has a thin shell encasing the volume mesh, the following block commands would be used:

```
Block 100 Volume 1
Block 100 Element Type Hex8
Block 200 Surface 1 To 6
Block 200 Element Type Shell4
Block 200 Attribute 0.01
Mesh Volume 1
Export Genesis 'block.g'
```

Which defines two element blocks (100 and 200). Element block 100 is composed of 8-node hexahedral elements and element block 200 is composed of 4-node shell elements on the surface of the block. The “thickness” of the shell elements is 0.01. The finite element code which reads the Genesis file (block.g) would refer to these blocks using the element block IDs 100 and 200. Note that the second line and the fourth line of the example are not required since both commands represent the default element type for the respective element blocks.

Surface Mesh Only

If a mesh containing only the surface of the block is desired, the first two lines of the example would be omitted and the **Mesh Volume 1** line would be changed to, for example, **Mesh Surface 1 To 6**.

Two-Dimensional Mesh

CUBIT also provides the capability of writing two-dimensional Genesis databases similar to FASTQ. The user *must* first assign the appropriate surfaces in the model to an element block. Then a **Quad*** type element may be specified for the element block. For example

```
Block 1 Surface 1 To 4
Block 1 Element Type Quad4
```

In this case, it is important for users to note that a two-dimensional Genesis database will result. In writing a two-dimensional Genesis database, CUBIT *ignores* all z-coordinate data. Therefore, the user must ensure that the Element Block is assigned to a planar surface lying in a plane parallel to the x-y plane. Currently, the **Quad*** element types are the only supported two-dimensional elements. Two-dimensional shell elements will be added in the near future if required.

▼ Boundary Conditions: Nodesets and Sidesets

Boundary conditions such as constraints and loads are applied to the finite element model through nodesets and sidesets. Nodesets can be created from groups of nodes categorized by their owning volumes, surfaces, or curves. Nodes can belong to more than one nodeset. Sidesets can be created from groups of element sides or faces categorized by their owning surfaces or curves. Element sides and faces can belong to more than one sideset. Nodesets and Sidesets can be viewed individually through CUBIT by employing the **Draw Nodeset** and **Draw Sideset** commands.

Nodesets and Sidesets may be assigned to the appropriate geometric entities in the model using the following commands in the command line:

Nodeset <nodeset_id> {Curve | Surface | Volume | Vertex} <range>

Sideset <sideset_id> {Curve | Surface} <range>

When using the GUI version of CUBIT, nodesets and sidesets are specified by accessing their respective dialog boxes from the **Constraints** menu. The Nodesets menu item will display the Nodeset Dialog and the Sidesets menu item will display the Sideset Dialog. The top portion of both of these are shown in ; the bottom portion is a standard picker window. The Geometry Type option menu can be set to **Body**, **Volume**, **Surface**, or **Curve**. The user-specified output identification number for the nodeset or sideset is entered in the **Nodeset ID** or **Sideset ID** text field. This is the number which will be used to identify this boundary condition in the exported EXODUSII file. The geometric entities to which this boundary condition is to be applied is then specified using the normal picking syntax.

Nodeset Associativity Data

Nodesets are also used to store geometry associativity data in the ExodusII file. This data can be used to associate the corresponding mesh to an existing geometry in a subsequent CUBIT session. This functionality can be used either to associate a previously-generated mesh with a geometry (“Mesh Importing and Duplicating” on page 127), or to associate a field function with a geometry for field function-based meshing “Adaptive Surface Meshing” on page 97).

The command syntax used to control whether or not associativity data is written to the ExodusII files is the following:

NodeSet Associativity { on | off }

Associativity data is stored in the ExodusII file in two locations. First, a nodeset is written for each piece of geometry (vertices, curves, etc) containing the nodes owned for that geometry. Then, the name of each geometry entity is associated with the corresponding nodeset by writing a property name and designating the corresponding nodeset as having that property. Nodeset numbers used for associativity nodesets are determined by adding a fixed base number (depending on the order of the geometric entity) to the geometric entity id number. The base

numbers for various orders of geometric entities are shown in Table 6-1. For example, nodes

Table 6-1. Nodeset id base numbers for geometric entities

Geometric Entity	Base Nodeset Id
Vertex	50000
Curve	40000
Surface	30000
Volume	20000

owned by curve number 26 would be stored in associativity nodeset 40026.

Instead of storing just the nodes owned by a particular entity, nodes for lower order entities are also stored. For example, the associativity nodeset for a surface would contain all nodes owned by that surface as well as the nodes on the bounding curves and vertices.

▼ Setting the Title

CUBIT will automatically generate a default title for the Genesis database. The default title has the form:

```
cubit(genesis_filename): date: time
```

The title can be changed using the command:

Title '<title_string>'

CUBIT's parser requires that strings be enclosed in single quotes, for example **'This is a string'**.

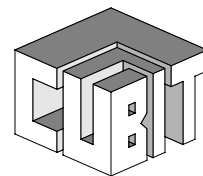
▼ Exporting the Finite Element Model

A Genesis database can be output using the **Export** option of the **File** Menu. Only Exodus II format is currently supported¹. A file can also be output using the following command:

Export Genesis '<filename>'

The Export Genesis command automatically creates a *unique* Element Block for every volume that is meshed (assuming the user has not entered any Block commands overriding this default behavior). Users can selectively control which blocks are output to the Genesis file since Element Blocks will *not* be created for any volumes that are not meshed.

1. Actually, there are two other formats provided for specialized applications. These formats are Xpatch and FRED. The command to create these file formats is **Export Xpatch|FRED '<filename>'**. If you need more information about these file formats, contact a CUBIT developer.



Appendix A: Command Index

▼ Command Syntax...139

▼ Commands...139

In this section the commands available in CUBIT are listed in alphabetical order. There may be more than one command available under a given command definition, since some commands can be executed singly or over a range of objects.

▼ Command Syntax

Each command will be first listed by a command heading, which will be phrased in standard terminology and will typically be very close to the computer syntax for the particular command. Underneath the command heading, each variation of the command will be listed, to document commands which can be applied to single objects as well as a range of objects. At the end of each command heading is a page number cross-reference to the location in the main document where the command is documented.

Command listings will ask for four types of command arguments: arguments include integers (6, 1, 3, etc., these are typically id's), reals (1.4, 2.35, etc., typically floating point quantities such as angles, spatial coordinates), strings (jjrome.jou, part.g, etc.), and logicals (1, 0, on, off, etc.).

Detailed descriptions of most commands and their usage can be found within the main sections of this document. A few of the special purpose commands are not documented elsewhere and only their syntax is shown below. These commands will be fully-documented after the functionality and generality are improved.

▼ Commands

At page 45

[View] At <X_coord> <Y_coord> <Z_coord> Animation Steps <number>

Block Attribute page 134

Block <block_id_range> Attribute <attribute>

Block Element Type page 134

Block <block_id_range> Element Type <element_type_name>

Block Geometry Type page 134

Block <id> {volume | surface | curve } <block_id_range>

Block Label

Block <block_id_range> Label { on | off }

Block Visibility

page 49

Block <block_id_range> Visibility { on | off }

Body Copy

page 72

Body <body_id_range> Copy [Move <x> <y> <z>]

Body <body_id_range> Copy [Reflect {x|y|z}]

Body <body_id_range> Copy [Reflect <x> <y> <z>]

Body <body_id_range> Copy [Rotate <angle> About {x|y|z}]

Body <body_id_range> Copy [Rotate <angle> About <x> <y> <z>]

Body <body_id_range> Copy [Scale <scale>]

Body Geometry Visibility

page 50

Body <body_id_range> Geometry { on | off }

Body <body_id_range> Geometry Visibility { on | off }

Body Interval

page 88

Body <body_id_range> Interval <interval>

Body <body_id_range> Interval {Hard|Soft|Default}

Body <body_id_range> Interval {Even|Odd}

Body Label

Body <body_id_range> Label {on | off | name | id | interval}

Body Mesh Visibility

page 50

Body <body_id_range> Mesh { on | off }

Body <body_id_range> Mesh Visibility { on | off }

Body Move

page 72

Body <body_id_range> [Copy] Move <x> <y> <z>

Body Reflect

page 73

Body <body_id_range> [Copy] Reflect {x|y|z}

Body <body_id_range> [Copy] Reflect<x> <y> <z>

Body Restore

page 73

Body <body_id_range> Restore

Body Rotate

page 73

Body <body_id_range> [Copy] Rotate <angle> About {x|y|z}

Body <body_id_range> [Copy] Rotate <angle> About <x> <y> <z>

Body Scale

page 73

Body <body_id_range> [Copy] Scale <scale>

Body Size

page 88

Body <body_id_range> Size [<size> | Smallest Curve]

Body Visibility


page 50

Body <body_id_range> { on | off }

Body <body_id_range> Visibility { on | off }

	BoundaryLayer	page 105
	BoundaryLayer <layer_id_range> First [Layer] <depth> Growth [Factor] <growth> Total Depth <depth> [Sublayer <depth>] BoundaryLayer <layer_id_range> First [Layer] <depth> Growth [Factor] <growth> Layers <count> [Sublayer <depth>]	
	BoundaryLayer Surface	page 105
	BoundaryLayer <layer_id> Curve <curve_id_range> Surface <surface_id>	
	Brick	page 67
	[Create] Brick Width <width> [Depth <depth> Height <height>]	
	Check	
	Check {bodies surfaces curves}	
	Color Background	page 43
	Color Background <color_name> Color Background <color_id>	
	Color Block	page 50
	Color Block <block_id_range> <color_name>	
	Color Body	page 50
	Color Body <body_id_range> <color_name> Color Body <body_id_range> <color_id>	
	Color Body Geometry	page 50
	Color Body <body_id> Geometry <color_name> Color Body <body_id> Geometry <color_id>	
	Color Body Mesh	page 50
	Color Body <body_id_range> Mesh <color_name> Color Body <body_id_range> Mesh <color_id>	
	Color Group	
	Color Group <group_id_range> <color_name> Color Group <group_id_range> <color_id>	
	Color Group Geometry	
	Color Group <group_id_range> Geometry <color_name> Color Group <group_id_range> Geometry <color_id>	
	Color Node	page 50
	Color Node <color>	
	Color NodeSet	page 50
	Color NodeSet <nodeset_id_range> <color>	
	Color SideSet	page 50
	Color SideSet <sideset_id_range> <color>	

	Color Surface	page 50
	Color Surface <surface_id_range> <color>	
	Color Surface <surface_id_range> <color_id>	
	Color Surface Geometry	page 50
	Color Surface <surface_id_range> Geometry <color>	
	Color Surface <surface_id_range> Geometry <color_id>	
	Color Surface Mesh	page 50
	Color Surface <surface_id_range> Mesh <color>	
	Color Surface <surface_id_range> Mesh <color_id>	
	Color Volume	page 50
	Color Volume <volume_id_range> <color>	
	Color Volume <volume_id_range> <color_id>	
	Color Volume Geometry	page 50
	Color Volume <volume_id_range> Geometry <color>	
	Color Volume <volume_id_range> Geometry <color_id>	
	Color Volume Mesh	page 50
	Color Volume <volume_id_range> Mesh <color>	
	Color Volume <volume_id_range> Mesh <color_id>	
	Color Sheet	page 50
	Color Sheet <sheet_id_range> <color>	
	Color Sheet <sheet_id_range> <color_id>	
	Comment	
	Comment 'text written to journal file'	
	Copy Mesh	page 128
	Copy Mesh Surface <surface_id> Onto Surface <surface_id>	
	Copy Mesh Volume <volume_id> Onto Volume <volume_id>	
	Create Brick	page 67
	[Create] Brick Width <width> [Depth <depth> Height <height>]	
	Create Cylinder	page 68
	[Create] Cylinder Height <height> Radius <radius>	
	[Create] Cylinder Height <height> Major Radius <radius> Minor Radius <radius>	
	Create Frustum	page 69
	[Create] Frustum Height <height> Major Radius <radius>	
	Minor Radius <radius> [Top <top_radius>]	
	[Create] Frustum Height <height> Radius <radius> [Top <top_radius>]	
	Create Prism	page 68
	[Create] Prism Height <height> Sides <sides> Major <radius>	
	Minor <radius>	
	[Create] Prism Height <height> Sides <sides> Radius <radius>	

	Create Pyramid	page 69
	[Create] Pyramid Height <height> Sides <sides> Major <radius> Minor <radius> Top <radius> [Create] Pyramid Height <height> Sides <sides> Radius <radius>	
	Create Sphere	page 69
	[Create] Sphere Radius <radius> [Create] Sphere Radius <radius> [Inner Radius <inner_radius>] [xpositive] [ypositive] [zpositive] [delete]	
	Create Torus	page 70
	[Create] Torus Rad1 <R1> Rad2 <R2>	
	Curve Interval	page 88
	Curve <curve_id_range> Interval <interval> Curve <curve_id_range> Interval {Hard Soft Default} Curve <curve_id_range> Interval {Even Odd}	
	Curve Label	
	Curve <curve_id_range> Label {on off name id interval}	
	Curve DicerSheet Interval	
	Curve DicerSheet Interval <interval>	
	Curve Reverse Bias	page 89
	Curve <curve_id_range> ReverseBias	
	Curve Scheme Curvature	
	Curve <curve_id_range> Scheme Curvature	
	Curve Scheme Bias	page 89
	Curve <curve_id_range> Scheme Bias Factor <growth_factor>	
	Curve Scheme Equal	page 89
	Curve <curve_id_range> Scheme Equal	
	Curve Scheme FeatureSize	page 89
	Curve <curve_id_range> Scheme FeatureSize	
	Curve Size	page 88
	Curve <curve_id_range> Size [<size> Smallest Curve]	
	Cylinder	page 68
	[Create] Cylinder Height <height> Radius <radius> [Create] Cylinder Height <height> Major Radius <radius> Minor Radius <radius>	
	Decompose	page 76
	Decompose <body_id> With <body_id>	
	Delete Body	
	Delete Body <body_id_range>	

Delete Face

page 127

Delete Face <face_id> Node1 <node1_id> Node2 <node2_id>

Delete Fine Mesh

Delete Fine Mesh {volume | surface | curve} <id_range> [propagate]

Delete Mesh

page 126

Delete Mesh

Delete Mesh Vertex <vertex_id_range>

Delete Mesh Curve <curve_id_range>

Delete Mesh Surface <surface_id_range>

Delete Mesh Volume <volume_id_range>

Delete Mesh Body <body_id_range>

Delete Mesh Group <group_id_range>

Delete Mesh {Group | Body | Volume | Surface | Curve} <id_range> Propagate

DicerSheet Interval

DicerSheet <id_range> Interval <interval>

DicerSheet Default Interval <interval>

Display

page 42

Display

Draw Block

page 48

Draw Block <block_id_range>

Draw Body

page 48

Draw Body <body_id_range>

Draw Curve

page 48

Draw Curve <curve_id_range>

Draw Edge

page 48

Draw Edge <edge_id_range>

Draw Face

page 48

Draw Face <face_id_range>

Draw Group

Draw Group <group_id_range>

Draw Hex

page 48

Draw Hex <hex_id_range>

**Draw Loop**

Undocumented

Draw Loop <loop_id_range>

Draw Node

page 48

Draw Node <node_id_range>

Draw NodeSet

page 48

Draw NodeSet <nodeset_id_range>

	Draw SideSet	page 48
	Draw SideSet <sideset_id_range>	
	Draw Surface	page 48
	Draw Surface <surface_id_range>	
	Draw Vertex	page 48
	Draw Vertex <vertex_id_range>	
	Draw Volume	page 48
	Draw Volume <volume_id_range>	
I	Echo	page 59
	[Set] Echo {on off}	
	Exit	page 41
	Exit	
	Quit	
I	Export	page 137
	Export Fred '<filename>'	
	Export Genesis '<filename>'	
	Export Xpatch '<filename>'	
	From	page 46
	[View] From <X_coord> <Y_coord> <Z_coord> Animation Steps <number>	
I	Frustum	page 69
	[Create] Frustum Height <height> Major Radius <radius> Minor Radius <radius> [Top <top_radius>]	
	[Create] Frustum Height <height> Radius <radius> [Top <top_radius>]	
I	Geometry Visibility	page 49
	Geometry { on off }	
	Geometry Visibility { on off }	
	Graphics Autocenter	page 44
	Graphics Autocenter {on off}	
	Graphics Autoclear	page 44
	Graphics Autoclear {on off }	
	Graphics Axis	page 44
	Graphics Axis {on off}	
	Graphics Border	page 44
	Graphics Border {on off}	
	Graphics Center	page 45
	Graphics Center	
	Graphics Clear	page 45
	Graphics Clear	

Graphics LineWidth	page 45
Graphics LineWidth <width>	
Graphics Mode	page 43
Graphics Mode FlatShade	
Graphics Mode HiddenLine	
Graphics Mode PolygonFill	
Graphics Mode Painters	
Graphics Mode SmoothShade	
Graphics Mode WireFrame	
Graphics Mode Dual	
Graphics Pan	
[Graphics] Pan {Left Right Up Down} <factor> Animation Steps <number>	
[Graphics] Pan Cursor Animation Steps <number>	
Graphics Perspective	page 47
Graphics Perspective {on off}	
Graphics Perspective Angle	page 47
Graphics Perspective Angle <view_angle_in_degrees>	
Graphics Status	
Graphics Status {on off}	
Graphics Text Size	page 51
Graphics Text Size <size_factor>	
Graphics Window	
Graphics Window Active <window_number>	
Graphics Window Create <window_number>	
Graphics Window Delete <window_number>	
Graphics WindowSize	page 43
Graphics WindowSize <width_in_pixels> <height_in_pixels>	
Graphics WindowSize Maximum	
Graphics Zoom	page 47
[Graphics] Zoom <X_min> <Y_min> <X_max> <Y_max>	
Animation Steps <number>	
[Graphics] Zoom Cursor Animation Steps <number>	
[Graphics] Zoom Reset	
[Graphics] Zoom Screen <Scale_Factor> Animation Steps <number>	
[Graphics] Zoom {group body volume surface curve vertex} <entity_id>	
Group	
Group 'group_name' Add <list_of_entity_ranges>	
Group <group_id> Add <list_of_entity_ranges>	
Group 'group_name' Remove <list_of_entity_ranges>	
Group <group_id> Remove <list_of_entity_ranges>	

Group Interval

Group <group_id_range> Interval <interval>
 Body <group_id_range> Interval {Hard|Soft|Default}
 Body <group_id_range> Interval {Even|Odd}

Group Geometry Visibility

Group <group_id_range> Geometry Visibility { on | off }

Group Mesh Visibility

Group <group_id_range> Mesh Visibility { on | off }

Group Label

Group <group_id_range> Label { on | off | interval | id | name }

Group Size

Group <group_id_range> Size [<size> | Smallest Curve

Group Sweep Volumes

page 65

Hardcopy

page 51

Hardcopy '<filename>' [encapsulated|postscript|eps] [color|monochrome]
 Hardcopy '<filename>' Pict [XSize <xsize>] [YSize <ysize>]

Help

page 61

Help
 Help <keyword>
 <keyword> Help

Highlight

Highlight {volume | Surface | Curve | Vertex | DicerSheet} <id_range>

**Hyperhelp**

page 61

Hyperhelp <keyword>
 <keyword> [<identifier>] Hyperhelp

Import Acis

page 70

Import Acis '<acis_filename>'

Import Fastq

page 70

Import Fastq '<fastq_filename>'

Import Geometry Mesh

page 118

Import Geometry Mesh '<exodusII_filename>' Block <block_number>

**Import Mesh**

page 128

Import Mesh <exodusII_filename> Block <block_id>
 Volume <volume_id>

**Import Sizing Function**

page 103

Import Sizing Function '<exodusII_filename>' Block <block_id>
 Variable '<variable_name>' Time <time_val>

Intersect

page 73

Intersect <body_id> With <body_id>

Journal

page 41

[Set] Journal {on | off}

Label

page 51

Label {on | off | interval | id | name }
 Label All { on | off | interval | id | name }
 Label Body { on | off | interval | id | name }
 Label Curve { on | off | interval | id | name }
 Label Edge { on | off | interval | id | name }
 Label Face { on | off }
 Label Geometry { on | off | interval | id | name }
 Label Group { on | off | interval | id | name }
 Label Hex { on | off }
 Label Mesh { on | off }
 Label Node { on | off }
 Label Surface { on | off | interval | id | name }
 Label Vertex { on | off | interval | id | name }
 Label Volume { on | off | interval | id | name }

List (Geometry/Mesh Related)

page 52

List Body <body_id_range> [{geometry|debug}]
 List Curve <curve_id_range> [{geometry|debug}]
 List DicerSheet [<id_range>]
 List Face <mesh_face_id_range>
 List Group <group_id_range> [{geometry|debug}]
 List Hex <hex_id_range>
 List Names [{Group|Body|Volume|Surface|Curve|Vertex}]
 List Node <node_id_range>
 List Surface <surface_id_range> [{geometry|debug}]
 List Totals
 List Model
 List Vertex <vertex_id_range> [{geometry|debug}]
 List Volume <volume_id_range> [{geometry|debug}]

List (Other)

page 58

List Debug
 List Echo
 List Information
 List Journal
 List Logging
 List Memory `<ClassName>`
 List Memory
 List Settings
 List View
 List Warning

	Merge	page 79
	Merge All	
	Merge All Curves	
	Merge All Surfaces	
	Merge Body <body_id> With Body <body_id>	
	Merge Body <body_id_range>	
	Merge Curve <curve_id> With <curve_id>	
	Merge Curve <curve_id_range>	
	Merge Surface <surface_id> With <surface_id>	
	Merge Surface <surface_id_range>	
	Mesh Body	page 124
	Mesh Body <body_id_range>	
	Mesh Body All	
	Mesh Curve	page 90
	Mesh Curve <curve_id_range>	
	Mesh Curve All	
	Mesh Group	
	Mesh Group <group_id_range>	
	Mesh Group All	
	Mesh Surface	page 105
	Mesh Surface <surface_id_range>	
	Mesh Surface All	
	Mesh Visibility	page 49
	Mesh { on off }	
	Mesh Visibility { on off }	
	Mesh Volume	page 124
	Mesh Volume <volume_id_range>	
	Mesh Volume <volume_id_range> Hexes <number_of_hexes>	
	Mesh Volume <volume_id_range> Levels <number_of_levels>	
	Mesh Volume All	
	Mouse	
	Mouse	
	Mouse2D	
	Mouse2D	
	Name	
	Name {Group Body Volume Surface Curve Vertex} <id> `entity_name`	
	Node Move	
	Node <node_id_range> Move <delta_x> <delta_y> <delta_z>	
	Node Visibility	page 49
	Node { on off }	
	Node Visibility { on off }	

	NodeSet Associativity	page 136
	NodeSet Associativity { on off }	
	NodeSet Curve	page 136
	NodeSet <nodeset_id> Curve <curve_id_range>	
	NodeSet Label	
	NodeSet <nodeset_id_range> Label {on off}	
	NodeSet Move	page 127
	NodeSet <nodeset_id> Move <X> <Y> <Z>	
	NodeSet <nodeset_id> Move To <X> <Y> <Z>	
	NodeSet Surface	page 136
	NodeSet <nodeset_id> Surface <surface_id_range>	
	NodeSet Vertex	page 136
	NodeSet <nodeset_id> Vertex <vertex_id_range>	
	NodeSet Visibility	page 49
	NodeSet { on off }	
	NodeSet Visibility { on off }	
	NodeSet <nodeset_id_range> { on off }	
	NodeSet <nodeset_id_range> Visibility { on off }	
	NodeSet Volume	page 136
	NodeSet <nodeset_id> Volume <volume_id_range>	
	Pan	
	[Graphics] Pan {Left Right Up Down} <factor> Animation Steps <number>	
	[Graphics] Pan Cursor Animation Steps <number>	
	Pause	page 42
	Pause	
	Pick	
	Pick {Curve Surface Volume Body DicerSheet [List] [Multiple]}	
	Pillow Volume	page 117
	Pillow Volume <volume_id>	
	Pillow Weave	page 118
	Pillow Weave {Automatic Buffer Create}	
	Pillow Sheet	page 118
	Pillow Sheet { Face <face_id1> [<face_id2>...] Hex <hex_id1> [<hex_id2>...] Cut Orient Verify}	
	Playback	page 42
	Playback '<journal_filename>'	
	Plot	
	Plot	

	Prism	page 68
	[Create] Prism Height <height> Sides <sides> Major <radius> Minor <radius>	
	[Create] Prism Height <height> Sides <sides> Radius <radius>	
	Pyramid	page 69
	[Create] Pyramid Height <height> Sides <sides> Major <radius> Minor <radius> Top <radius>	
	[Create] Pyramid Height <height> Sides <sides> Radius <radius>	
	Quality	
	Quality <entity_list> [Global]	
	Quality <entity_list> [Global] Display Draw `Metric Name`	
	Quit	page 41
	Quit	
	Exit	
	Record	page 42
	Record '<journal_filename>'	
	Record Stop	page 42
	Record Stop	
	Replace Mesh	
	Replace Mesh {group volume surface} <id_range>	
	Reset	page 41
	Reset	
	Reset Blocks	
	Reset Genesis	
	Reset Nodesets	
	Reset SideSets	
	Rotate	page 46
	Rotate <degree> About [Screen World Camera] {x y z} Animation Steps <number>	
	Rotate <degree> About Curve <curve_id> Animation Steps <number>	
	Rotate <degree> About Vertex <axis_start_vertex_id> Vertex <axis_end_vertex_id> Animation Steps <number>	
	Set	page 59
	[Set] Debug <flag_id> {on off} [{File 'filename' Terminal}]	
	[Set] Echo {on off}	
	[Set] Info {on off}	
	[Set] Journal {on off}	
	[Set] Logging {on off} [{File 'filename' Terminal}]	
	[Set] Warning {on off}	

Set	page 59
Set Uncross On	
Set Weavedata On	
Set Step On	
Set Arrangement {On Off}	
Set Sheets {On Off}	
Set	page 115
Set Windows On	
Set	page 117
Set Superdrive {On Off}	
Sheet Visibility	page 49
Sheet <sheet_id_range> Visibility { on off }	
SideSet Curve	page 136
SideSet <sideset_id> Curve <curve_id_range>	
SideSet Label	
SideSet <sideset_id_range> Label { on off }	
SideSet Surface	page 136
SideSet <sideset_id> Surface <surface_id_range>	
SideSet Visibility	page 49
SideSet { on off }	
SideSet Visibility { on off }	
SideSet <sideset_id_range> { on off }	
SideSet <sideset_id_range> Visibility { on off }	
Smooth Group	
Smooth Group <group_id_range>	
Smooth Surface	page 125
Smooth Surface <surface_id_range> [Global]	
Smooth Volume	page 126
Smooth Volume <volume_id_range>	
Sphere	page 69
[Create] Sphere Radius <radius>	
[Create] Sphere Radius <radius> [Inner Radius <inner_radius>]	
[xpositive] [ypositive] [zpositive] [delete]	
Subtract	page 74
Subtract <body_id> From <body_id>	
Surface Angle	
Surface <surface_id_range> Angle <angle_degrees>	

Surface Default Scheme

Surface Default Scheme {map | pave | submap | triangle}

Surface DicerSheet Interval

Surface <id_range> DicerSheet Interval <interval>

Surface Geometry Visibility

page 50

Surface <surface_id_range> Geometry Visibility { on | off }

Surface Interval

page 88

Surface <surface_id_range> Interval <intervals>

Surface <surface_id_range> Interval {Hard|Soft|Default}

Surface <surface_id_range> Interval {Even|Odd}

Surface Label

Surface <surface_id_range> Label {on | off | name | id | interval}

Surface Mesh Visibility

page 50

Surface <surface_id_range> Mesh Visibility { on | off }

Surface Periodic Interval

Surface <surface_id_range> Periodiic Interval <intervals>

Surface Scheme Circle

Surface <surface_id_range> Scheme Circle [Interval <intervals>]

**Surface Scheme Curvature**

page 94

Surface <surface_id_range> Scheme Curvature

Surface Scheme Dice

Surface <surface_id_range> Scheme Dice

Surface Scheme Map

page 94

Surface <surface_id_range> Scheme Map

Surface Scheme Morph

Surface <surface_id_range> Scheme Morph

Surface <surface_id_range> Scheme Morph {Source Node <id> Target Node <id>}

Surface <surface_id_range> Scheme Morph {Source Edge <id> Target Edge <id>}

Surface <surface_id_range> Scheme Morph

{Source Vertex <id> Target Vertex <id>}

Surface <surface_id_range> Scheme Morph

{Source Curve <id> Target Curve <id>}

**Surface Scheme Pave**

page 94

Surface <surface_id_range> Scheme Pave

Surface Scheme Submap

page 94

Surface <surface_id_range> Scheme Submap

Surface Scheme Triangle

page 94

Surface <surface_id_range> Scheme Triangle

Surface Scheme TriMap

Surface <surface_id_range> Scheme TriMap

Surface Scheme TriPave

Surface <surface_id_range> Scheme TriPave

Surface Size

page 88

Surface <surface_id_range> Size [<intervals>| Smallest Curve]

Surface Sizing Function

page 98, 103

Surface < id > Sizing Function Type { Curvature | Linear | Interval | Inverse |
Test | Exodus } [Min <min_val> Max <max_val>]

Surface Smooth Scheme

page 125

Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]

Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]

Weight Jacobian

Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]

Weight Area

Surface <surface_id_range> Smooth Scheme Equipotential [Fixed]

Weight Inverse [Area]

Surface <surface_id_range> Smooth Scheme Laplacian [Fixed]

Surface <surface_id_range> Smooth Scheme Centroid Area Pull [Fixed]

Surface Submap Smooth

Surface <id> SubMap Smooth <on|off>

Surface Vertex Types

Surface <surface_id> Vertex <vertex_id> Type {end|side|corner|reversal}

Surface <surface_id> Vertex <vertex_id> Type {triangle|nottriangle}

Surface Visibility

page 49

Surface <surface_id_range> { on | off }

Surface <surface_id_range> Visibility { on | off }

Title

page 137

Title '<title>'

Torus

page 70

[Create] Torus Major [Radius] <R1> Minor [Radius] <R2>

Unite

page 74

Unite <body_id> With <body_id>

Unite Body <body_id_list> [All]

Up

page 46

[View] Up <X_coord> <Y_coord> <Z_coord> Animation Steps <number>

Version

page 41

Version

Vertex

Vertex <vertex_id_range> {on | off}

Vertex Label

Vertex <vertex_id_range> Label {on | off | name | id | interval}

Vertex Visibility

page 49

Vertex { on | off }

Vertex Visibility { on | off }

Video

Video { on | off }

Video Initialize

page 51

Video Initialize [<number_of_frames>]

Video Initialize 'base_filename' pict [Xsize <xsize>] [Ysize <ysize>]

Video Snap

page 51

Video Snap

View At

page 45

[View] At <X_coord> <Y_coord> <Z_coord> Animation Steps <number>

View From

page 46

[View] From <X_coord> <Y_coord> <Z_coord> Animation Steps <number>

View List

page 47

View List

View Reset

View Reset

View Up

page 46

[View] Up <X_coord> <Y_coord> <Z_coord> Animation Steps <number>

Volume Default Scheme

Volume Default Scheme {map | submap | plaster | weave}

Volume DicerSheet Interval

Volume <volume_id_range> DicerSheet Interval <interval>

Volume Geometry Visibility

page 50

Volume <volume_id_range> Geom { on | off }

Volume <volume_id_range> Geometry Visibility { on | off }

Volume Interval

page 88

Volume <volume_id_range> Interval <intervals>

Volume <volume_id_range> Interval {Hard|Soft|Default}

Volume <volume_id_range> Interval {Even|Odd}

Volume Label

Volume <volume_id_range> Label {on | off | name | id | interval}

Volume Mesh Visibility

page 50

Volume <volume_id_range> Mesh { on | off }

Volume <volume_id_range> Mesh Visibility { on | off }

	Volume Scheme Curvature	
	Volume <volume_id_range> Scheme Curvature	
	Volume Scheme Dice	
	Volume <volume_id_range> Scheme Dice	
	Volume Scheme Map	page 106
	Volume <volume_id_range> Scheme Map	
	Volume Scheme Submap	page 106
	Volume <volume_id_range> Scheme Submap	
	Volume Scheme Plaster	page 106
	Volume <volume_id_range> Scheme Plaster	
	Volume Scheme Project	page 106
	Volume <volume_id> Scheme Project Source Surface <surface_id_list>	
	Target Surface <surface_id>	
	Volume Scheme Rotate	page 106
	Volume <volume_id> Scheme Rotate Source Surface <surface_id_list>	
	Target Surface <surface_id>	
	Volume Scheme Translate	page 106
	Volume <volume_id> Scheme Translate	
	Source Surface <surface_id_list> Target Surface <surface_id>	
	Volume Scheme Weave	page 106
	Volume <volume_id_range> Scheme Weave	
	Volume Size	page 88
	Volume <volume_id_range> Size [<interval_size> Smallest Curve]	
	Volume Smooth Scheme	page 125
	Volume <volume_id_range> Smooth Scheme Laplacian	
	Volume <volume_id_range> Smooth Scheme Equipotential [Fixed]	
	Volume Visibility	page 50
	Volume <volume_id_range> { on off }	
	Volume <volume_id_range> Visibility { on off }	
	WebCut Body	page 76
	Webcut Body <body_id> Face <face_id> [Vector <from_vertex> <to_vertex>]	
	Webcut Body <body_id> Vertex <vertex_1> Vertex <vertex_2>	
	Vertex <vertex_3> [Vector <from_vertex> <to_vertex>]	
	Weight Hexes	page 126
	Weight Hexes Surface <range> <weight>	

Zoom

page 47

[Graphics] Zoom <X_min> <Y_min> <X_max> <Y_max>

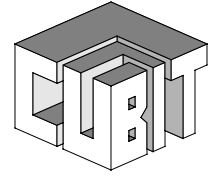
Animation Steps <number>

[Graphics] Zoom Cursor Animation Step <number>

[Graphics] Zoom Screen <Scale_Factor> Animation Steps <number>

[Graphics] Zoom Reset

[Graphics] Zoom {group|body|volume|surface|curve|vertex} <entity_id>



Appendix B: Examples

- ▼ General Comments...159
- ▼ Simple Internal Geometry Generation...160
 - ▼ Octant of Sphere...161
 - ▼ Airfoil...163
 - ▼ The Box Beam...164
- ▼ Thunderbird 3D Shell...167
- ▼ Assembly Components...170

The purpose of this Appendix is to demonstrate the capabilities of CUBIT for finite element mesh generation as well as provide a few examples on the use of CUBIT. Some examples also demonstrate the use of the ACIS test harness as well as other related programs. This Appendix is not intended to be a step-by-step tutorial.

▼ General Comments

CUBIT is based upon the ACIS solid modeling kernel. Solid models can be created within CUBIT or imported in the form of an ACIS geometry file¹. Current means of generating ACIS solid models external to CUBIT include:

- ACIS Test Harness
- FASTQ via the FASTQ to ACIS translator *fsqacs*²
- Aries[®] ConceptStation
- PRO/Engineer via a PRO/Engineer to ACIS translator

These examples show model construction using internal CUBIT geometry creation, the ACIS Test Harness, and the FASTQ translator. Those methods provide the capability of semi-automatically generating a mesh in batch mode in much the same manner as FASTQ [5], GEN3D [10], GREPOS [11], and GJOIN [12].

A CUBIT journal file is included for the examples shown in this appendix. ACIS journal files are also provided for the examples that require geometry generated by ACIS. The user can

-
1. ACIS typically adds the filename suffix “.sat” to the output files it writes in text format; therefore, these files are typically referred to as “.sat” files or “ACIS .sat” files. CUBIT cannot read binary ACIS files.
 2. The *fsqacs* users manual is reproduced in Appendix C, “Fsqacs: A FASTQ to ACIS Command Interpreter” on page 163

reproduce the examples interactively by simply entering each of the lines in the journal files as commands to CUBIT or ACIS. The examples assume that the command line version of CUBIT will be used. The examples can also be run using the Graphical User Interface version of CUBIT; however, the details for doing this are not given in this appendix. The journal files included in the example are also distributed with CUBIT and they may be executed using the **Playback 'filename'** command.

The examples in this appendix each cover several of CUBIT's mesh generation capabilities. The CUBIT features exercised by each example are shown in Table B-1.

Examples	Geometry Features				Surface Meshing Features						Volume Meshing Features			
	Primitives	Booleans	Geometry Editing	Geometry Consolidation	Curve Bias	Curvature-based	Mapping	Paving	Triangle Tool	Boundary Layer Tool	Project	Translate/Rotate	Mapping	Plastering
Internal Geometry	x	x						x				x		
Sphere Octant	x	x	x	x				x	x		x	x		
Airfoil					x			x		x				
Box Beam				x			x							
Thunderbird								x						
Assembly Components				x				x			x			

Table B-1 CUBIT Features Exercised by Examples.

▼ Simple Internal Geometry Generation

This simple example demonstrates the use of the internal geometry generation capability within CUBIT to generate a mesh on a perforated block. The geometry for this case is a block with a cylindrical hole in the center. It illustrates the **brick**, **cylinder**, **subtract**, **pave**, and **translate** commands and boolean operations. The geometry to be generated is shown in Figure B-1. This figure also shows the curve and surface labels specified in the CUBIT journal file. The final meshed body is shown in Figure B-2. The CUBIT journal file is:

Internal Geometry Generation Example

```
Brick Width 10. Depth 10. Height 10. # Create Cube
Cylinder Height 12. Radius 3. # Create cylinder through Cube
View From 3 4 5 # Update viewing position
Display
```

```

Subtract 2 From 1      # Remove cylinder from cube—create hole
Display
Body 3 Size 1.0        # Default element size for model
Surface 10 Interval 10  # Change intervals on cylinder surface
Curve 15 to 16 Interval 20 # Change intervals around cyl. circ.
Surface 11 Scheme Pave      # Front surface paved
Volume 3 Scheme Translate Source 11 Target 12      #Remainder
                                     # of block will be meshed by
                                     # translating front surface to back surface
Mesh Volume 3          # Create the mesh
Graphics Mode Hiddenline
Display                # Hiddenline view of cube (Figure B-2)

```

The first two lines create a 10 unit cube centered at the origin and a cylinder with radius 3 units and height of 12 units also centered at the origin. The cylinder height is arbitrary as long as it is greater than the height of the brick. The **subtract** command then performs the boolean by subtracting the cylinder (body 2) from the block (body 1) to create the final geometry (body 3). The remainder of the commands simply assign the desired number of intervals and then generate the mesh. Note that since the cylindrical hole is a “periodic surface,” there are no edges joining the two curves so the number of intervals along its axis must be set by the surface interval command. The steps required for generating this geometry and mesh using the Graphical User Interface are given in the Tutorial in Chapter 2.

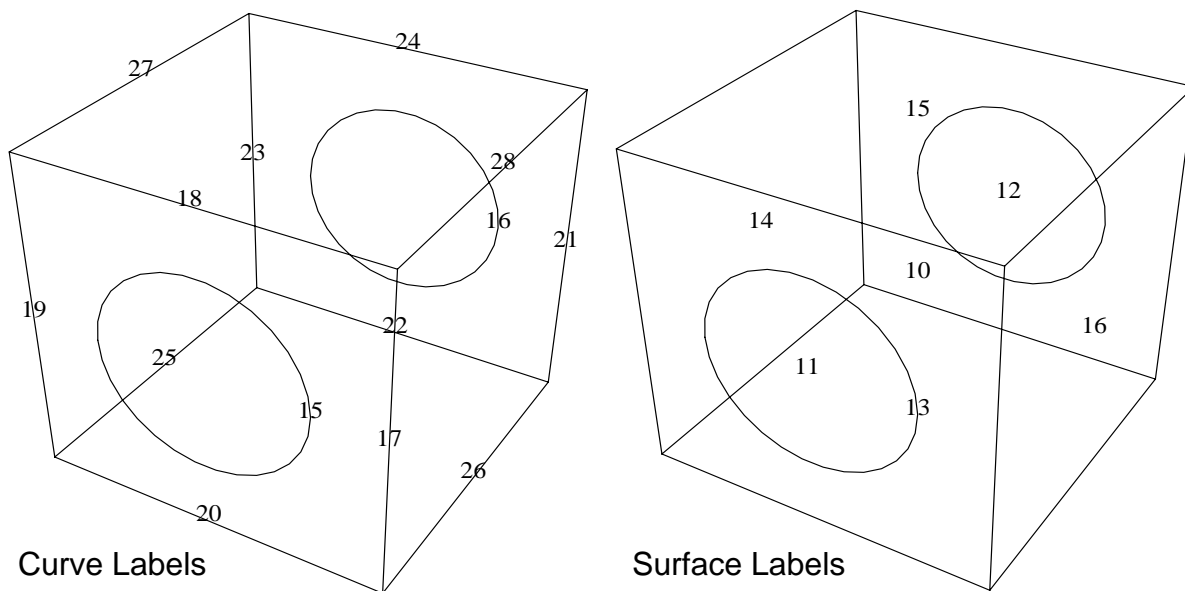


Figure B-1 Geometry for Cube with Cylindrical Hole

▼ Octant of Sphere

This example also illustrates the internal geometry generation capabilities of CUBIT to generate an octant of a sphere. The procedure used is to generate the octant by intersecting a brick with a sphere. The octant is then split into two pieces—a central “core” and an outer “peel” which are both meshable using the sweeping schemes. This example uses the **sphere**, **brick**, **cylinder**, **intersect**, **copy**, **subtract**, **merge**, **pave**, **project**, and **rotate** commands.

The following annotated CUBIT journal file will generate the mesh shown in Figure B-3.

```

Sphere Radius 10.          #Generate Sphere (Body 1)
Brick Width 12 Depth 12 Height 12      #Generate Cube (Body 2)

```

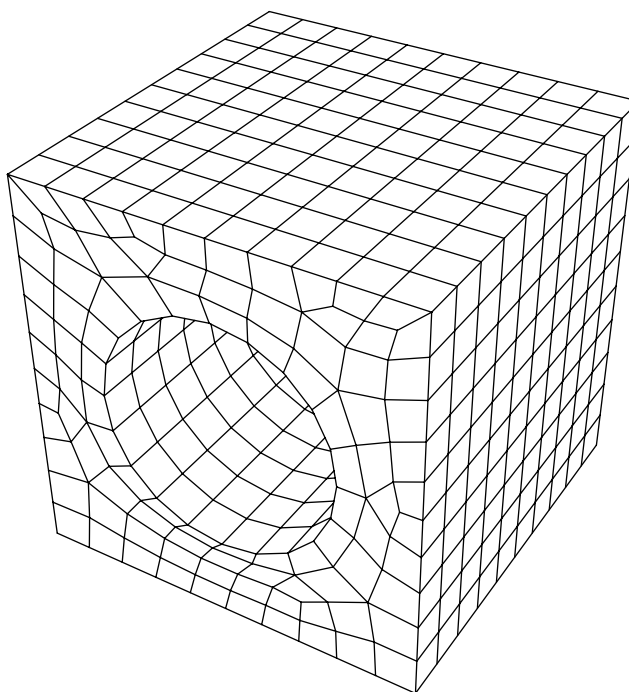


Figure B-2 Generated Mesh for Cube with Cylindrical Hole

```

Body 2 Move 6. 6. 6.                                #Move Cube to Enclose Octant
Graphics Mode SmoothShade                            #Only way to see a sphere
Display
Intersect 1 With 2                                    #Generate Octant (Body 3)
Display
Cylinder Height 22 Radius 3                          #Generate Cylinder (Body 4)
Body 3 To 5 Copy                                     #Copy Octant (Body 5)
                                                    #and Cylinder (Body 6)
                                                    #and another octant (Body 7)
                                                    #Create Core (Body 8)

Intersect 4 With 5
View From 1 2 3
Intersect 3 With 6                                    #Create Another Core (Body 9)
Subtract 8 From 7                                     #Create Peel (Body 10)
Merge All                                             #Coalesce Redundant Surfaces
#
# End of Geometry Generation.
# "Core" is volume/body 9
# "Shell" is volume/body 10
#
volume 9 Size 0.5
Surface 33 Scheme Pave                                #Pave end of core
Mesh Surface 33
volume 9 Scheme Project Source 33 Target 31 #Generate core mesh
curve 44 interval 10                                #set compatible target intervals
Mesh volume 9
Display                                              #Make sure it's there
#
volume 10 Size 0.5                                    #Make intervals agree for rotate
Surface 37 Scheme Pave                                #Pave face of peel
Mesh Surface 37
volume 10 Scheme Rotate Source 37 Target 40 #Generate Peel Mesh
Mesh volume 10
Display
Export Genesis 'Octant.gen'                          #Write out the mesh

```

If the generated mesh should consist of one material, the **block** command could be used to merge the peel and core into a single material block. Note that during a boolean operation (unite, intersect, and subtract), the bodies used in that boolean are destroyed so it is sometimes

necessary to create extra copies of a body prior to using them in a boolean operation. Also,

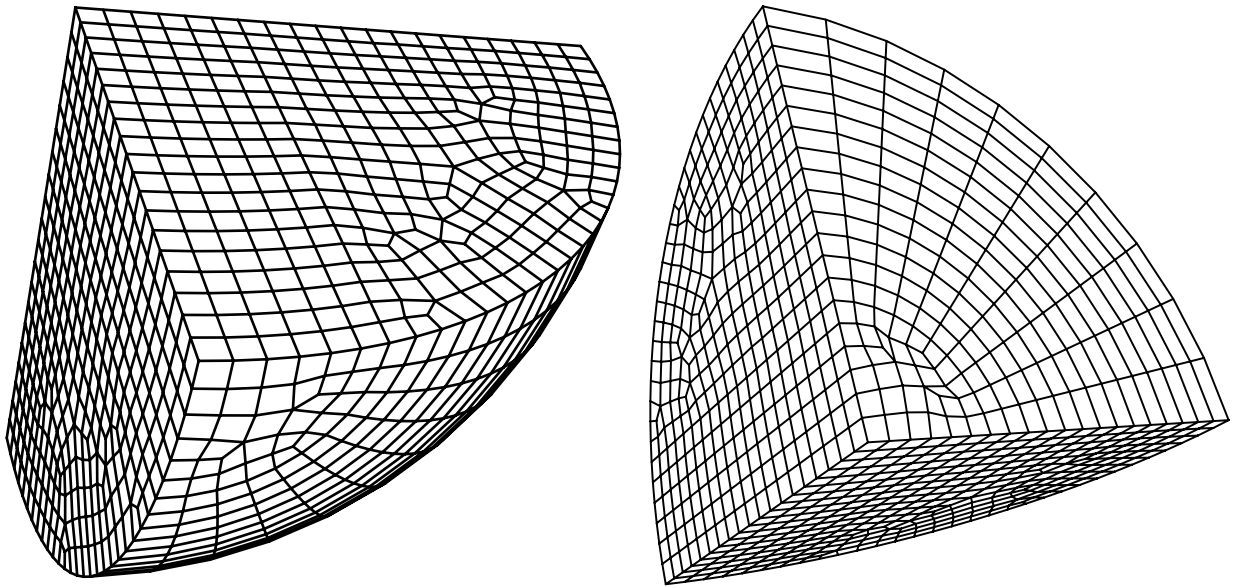


Figure B-3 Generated Mesh for Octant of Sphere

during boolean operations, many bodies are created and deleted and it is difficult to remember which bodies exist at certain times¹. It is recommended that comments be added to the journal file to make it easier to determine what is being done in the file.

▼ Airfoil

A simple two-dimensional airfoil is used in this example to demonstrate the use of the boundary layer tool and paving. The commands used to generate the geometry for this problem, using the ACIS Test Harness, are not included here. This example uses the **curve bias**, **boundarylayer** and **pave** commands. The CUBIT commands used to mesh this problem are:

```
# File: foil.jou
#
# Air Foil Example
#
journal off
Import Acis 'foil.sat'
View From 100 0 0 Up 0 0 1          # Set up View
AutoCenter On
Display
Volume 1 Interval 14                 # Set Meshing Parameters
Curve 4 Interval 24
Curve 2 Interval 24
Curve 5 To 6 Interval 18
Curve 6 Bias 1.1
Curve 5 Bias 0.909
BoundaryLayer 1 First Layer 0.5 Growth 1.3
```

1. The CUBIT Developers are very much aware of the problems this causes during the generation of complicated meshes and are implementing methods to permit user-defined naming of bodies and volumes. This capability relies on the persistent ID concept recently added to ACIS.

```

BoundaryLayer 1 Surface 1 Curve 5 to 6
Surface 1 Scheme Pave
Mesh Surface 1
Display
Graphics zoom .25 .4 .45 .6
geometry visibility off
display
geometry visibility on
display
# Create the Mesh

```

The mesh generated by these commands is shown in Figure B-4. In this example, curves 5 and 6 (the curves used to define the shape of the airfoil) use biased interval spacing to place more elements towards the front of the airfoil. A boundary layer is designated on either side of the airfoil, which produces elements with high aspect ratios for several layers around the airfoil. The parameters to the **boundarylayer** command specify the depth of the first and second rows of elements, with the boundary layer growth factor inferred from these data. The paving scheme generates the mesh outside the boundary layer.

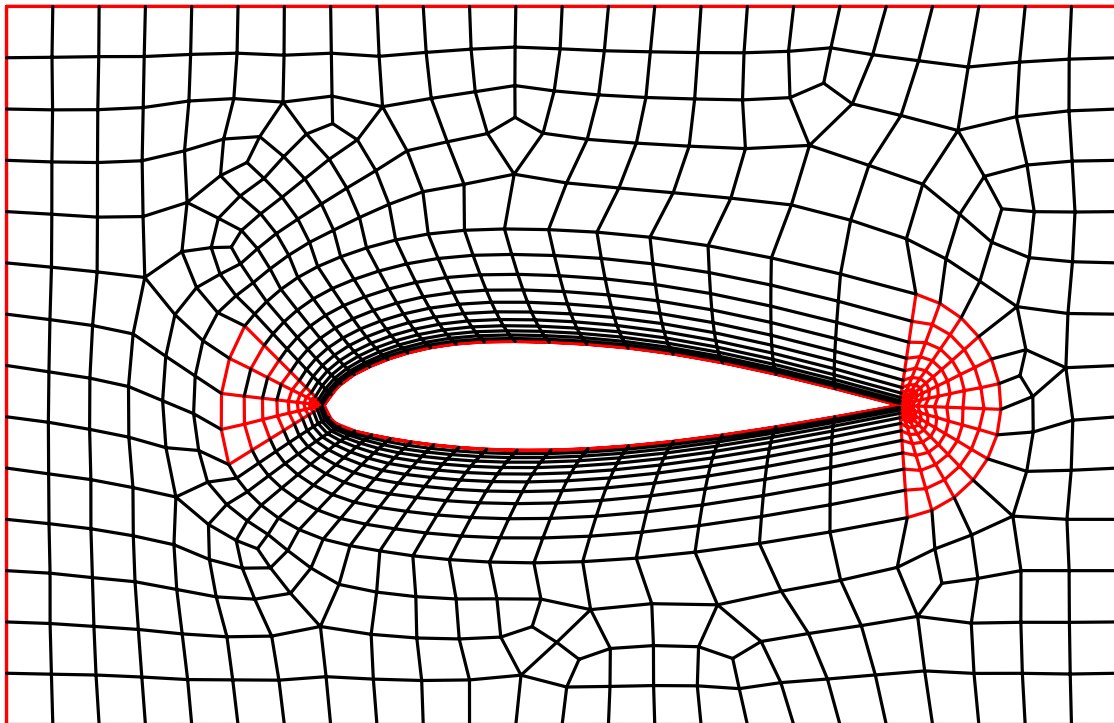


Figure B-4 Airfoil mesh generated using the boundary layer tool and paving.

▼ The Box Beam

A simple example using ACIS/CUBIT is the box beam buckling problem shown in Figure B-5. A description of an analysis which uses this type of mesh is found in Reference [15]. This example uses the **merge**, **nodeset** and **block** commands and the mapping mesh generation scheme.

The input file for the ACIS Test Harness for the box beam example is¹:

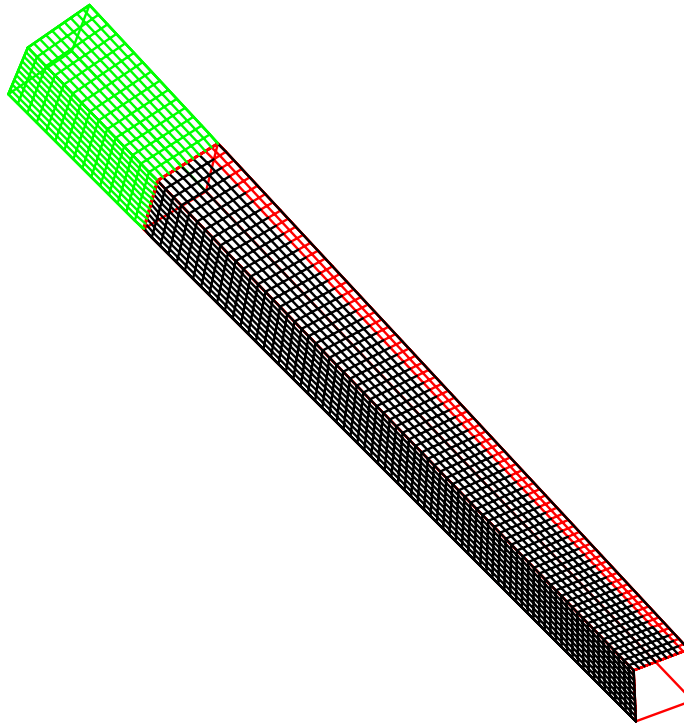


Figure B-5 Box Beam example

```
# File: boxBeam.mon
# Side = {Side = 1.75}
# Height = {Height = 12.0}
# Upper = {Upper = 2.0}

block lowerSection width {Side/2.0} depth {Side/2.0} height
{Height - Upper}
block upperSection width {Side/2.0} depth {Side/2.0} height
{Upper}

move lowerSection {Side/4.0} {Side/4.0} {(Height - Upper)/2.0}
move upperSection {Side/4.0} {Side/4.0} {Upper/2.0 + Height -
Upper}

group lowerSection upperSection as boxBeam

save boxBeam to boxBeam.sat
```

In this example, it is assumed that subsequent analyses will take advantage of the problem symmetry and therefore only one-quarter of the box beam will be meshed. It is worth noting that there are a variety of ways to construct a solid model for this problem; however, experience thus far with ACIS and CUBIT indicates that the easiest way to model the box beam is to use ACIS block primitives¹. Even though subsequent meshing will only be performed on the faces of the solid model, the entire 3D body is saved as an ACIS.sat file. The CUBIT journal file for the box beam example is:

-
1. This file must be preprocessed by Aprepro prior to being input to the ACIS Test Harness.
 1. This geometry can also be generated using the internal CUBIT Brick primitive.

File: boxBeam.jou - ?? no such file

```

# Thickness = {Thickness = 0.06}
# Crease = {Crease = 0.01}
# XYInts = {XYInts = 10}
# ZInts = {ZInts = 90}
# UpperInts = {UpperInts = 15}

Import Acis 'boxBeam.sat'
#Display

Merge All
Label Surface on
#Display

Label Curve on
#Display

Curve 1 To 8 Interval {XYInts}
Curve 13 To 16 Interval {XYInts}

Curve 9 To 12 Interval {ZInts-UpperInts}
Curve 21 To 24 Interval {UpperInts}

?? the following fail ??
Mesh Surface 3
Mesh Surface 6
Mesh Surface 9
Mesh Surface 12

NodeSet 1 Curve 1
NodeSet 2 Curve 4

NodeSet 1 Move {-Crease} 0 0
NodeSet 2 Move 0 {Crease} 0

Block 2 Surface 3
Block 2 Surface 6

Block 1 Surface 9
Block 1 Surface 12

Block 1 To 2 Attribute {Thickness}

Export Genesis 'boxBeam.exoII'
Quit

```

Commands worth noting in the CUBIT journal file include:

- **Block, Block Attribute** Allows the user to specify that shell elements for the surfaces of the solid model are to be written to the output (EXODUSII) database, and that shell elements be given a thickness attribute. This is necessary since CUBIT defaults to three-dimensional hexahedral meshing of solid model volumes.
- **NodeSet Move** Allows the user to actually move the specified nodes by a vector (Δx , Δy , Δz). This is advantageous for the buckling problem, since the numerical simulation requires a small “crease” in the beam in order to perform well.
- **Merge** Allows the user to combine geometric features (e.g. edges and surfaces).

Other commands in the journal file should be straightforward. Since the problem is sufficiently simple to mesh using a mapping transformation, specification of a meshing “scheme” is unnecessary (mapping is the default in CUBIT).

Finally, note that both the ACIS monitor file (**boxBeam.mon**) and the CUBIT journal file (**boxBeam.jou**) contain macros that are evaluated using Aprepro. The *makefile* used to semi-automatically generate the mesh is given below:

File: Makefile

```
boxBeam.g:boxBeam.exoII
    exo2exol boxBeam.exoII boxBeam.g

boxBeam.exoII:boxBeam.sat boxBeam.jou
    aprepro boxBeam.jou | cubitb
    rm cubit.jou

boxBeam.sat: boxBeam.mon
    aprepro boxBeam.mon | acis
    rm wjbohnhl.*

clean:
    @-rm *.sat *.exoII *.g
```

While this particular example is a trivial use of the software, it does serve to demonstrate a few of the capabilities offered by ACIS and CUBIT.

▼ Thunderbird 3D Shell

This example is the three-dimensional paving of a shell shown in Figure B-6. The 2D wireframe geometry of the thunderbird is given by the following FASTQ file:

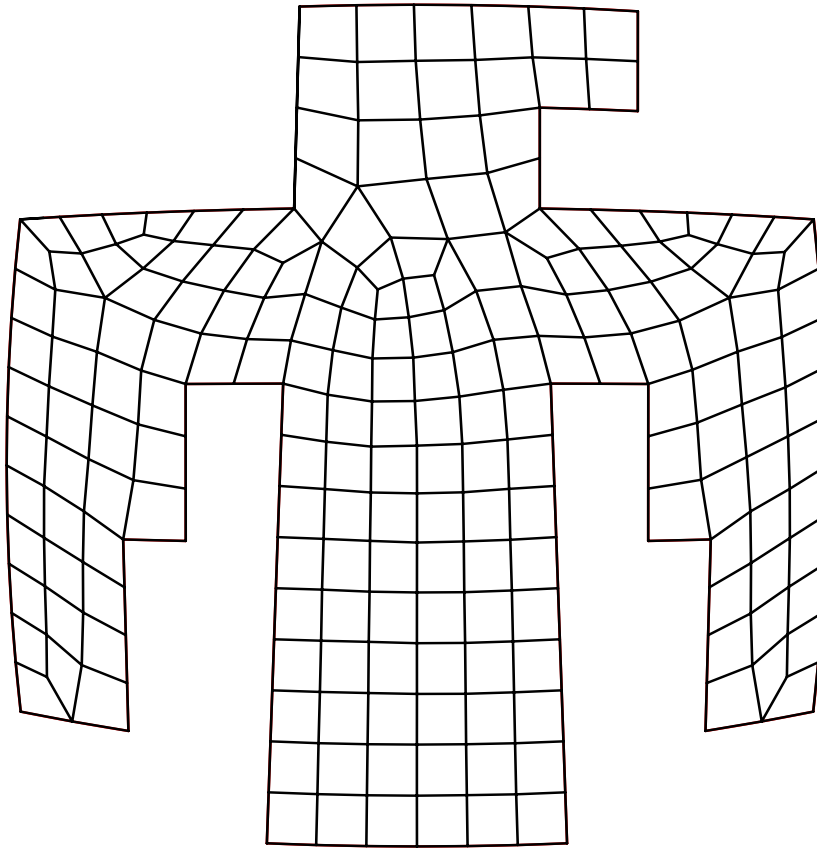


Figure B-6 Sandia Thunderbird 3D shell - ?? picture is different now!! ??

#File: tbird.fsq

```

TITLE
MESH OF SANDIA THUNDERBIRD

$ block {e = .2} int= {isq = 20}
$ number of elements in block thick {iblkt = 5} block thickness
{blkt=.2}
$ block angle {angle=15}
$ magnification factor = {magnificationFactor=1.0}
$ bird {bthick = .018} {ithick = 3} {idepth = 20}
$ {pi = 3.14159265359} {rad=magnificationFactor/pi} {bdepth=1.}
$ preferred normalized element size = {elementSize=0.06}
$ number of intervals along outside edges =
$ {border_int=5} {corner_int=10} {side_int=20}
$ {outsideIntervals= 2*corner_int+side_int}
$ {boxTop=.2} {topIntervals = 8}

$ {insideCurveInt=8}

$ {MAG=magnificationFactor/3.0}

$ {middleInside=MAG*0.97}
$ {xCurveStartInside=MAG*0.60}
$ {yCurveStartInside=MAG*0.93}
$ {curveMiddleInside=MAG*0.81}

$ {xCurveStartOutside=MAG*0.75}
$ {yCurveStartOutside=MAG*1.17}
$ {middleOutside=MAG*1.20}
$ {curveMiddleOutside=MAG*1.01}
$ {boundingBox = MAG*1.5}

$ Thunderbird Coordinates

POINT 1 {MAG*-.40} {MAG*.78}
POINT 2 {MAG*-.40} {MAG*.59}
POINT 3 {MAG*-.22} {MAG*.59}
POINT 4 {MAG*-.22} {MAG*.40}
POINT 5 {MAG*-.75} {MAG*.40}
POINT 6 {MAG*-.78} {MAG*-.09}
POINT 7 {MAG*-.75} {MAG*-.58}
POINT 8 {MAG*-.53} {MAG*-.60}
POINT 9 {MAG*-.54} {MAG*-.23}
POINT 10 {MAG*-.42} {MAG*-.23}
POINT 11 {MAG*-.42} {MAG*.07}
POINT 12 {MAG*-.24} {MAG*.07}
POINT 13 {MAG*-.27} {MAG*-.80}
POINT 14 {MAG*.27} {MAG*-.80}
POINT 15 {MAG*.24} {MAG*.07}
POINT 16 {MAG*.42} {MAG*.07}
POINT 17 {MAG*.42} {MAG*-.23}
POINT 18 {MAG*.54} {MAG*-.23}
POINT 19 {MAG*.53} {MAG*-.60}
POINT 20 {MAG*.75} {MAG*-.58}
POINT 21 {MAG*.78} {MAG*-.09}
POINT 22 {MAG*.75} {MAG*.40}
POINT 23 {MAG*.22} {MAG*.40}
POINT 24 {MAG*.21} {MAG*.78}
POINT 25 {MAG*0.0} {MAG*.80}

$ lines for Tbird

LINE 1 STR 1 2
LINE 2 STR 2 3
LINE 3 STR 3 4
LINE 4 STR 4 5
LINE 5 CIRM 5 7 6
LINE 6 STR 7 8
LINE 7 STR 8 9
LINE 8 STR 9 10

```

```

LINE 9 STR 10 11
LINE 10 STR 11 12
LINE 11 STR 12 13
LINE 12 STR 13 14
LINE 13 STR 14 15
LINE 14 STR 15 16
LINE 15 STR 16 17
LINE 16 STR 17 18
LINE 17 STR 18 19
LINE 18 STR 19 20
LINE 19 CIRM 20 22 21
LINE 20 STR 22 23
LINE 21 STR 23 24
LINE 22 STR 24 1 0 7 1.0

$ REGIONS

SIZE {elementSize*MAG}

REGION 1 1 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 -11 -12 -13 -14 -15 *
      -16 -17 -18 -19 -20 -21 -22

SCHEME 0 X
BODY 1
EXIT

```

A command interpreter program, **fsqacs**¹, has been developed to convert FASTQ geometry commands to equivalent ACIS Test Harness commands (outputs an ACIS monitor file). Note, **fsqacs** ignores any meshing information in the FASTQ file since there is currently no means of passing the mesh parameters through the ACIS solid modeler to the CUBIT session. It should be noted that the 2D wireframe geometry can be directly constructed using wires in the ACIS Test Harness; however, there may be instances when it is more convenient to use the command interpreter.

After executing **fsqacs**, the resulting ACIS monitor file may be included in a subsequent ACIS session by simply using the **include** command as illustrated by the following file:

#File: tbird3d.mon

```

include tbird.acs
roll
view scale 200
#draw
cylinder cyll height 1.25 radius 0.5
rotate cyll by 90 about x
#draw
sweep wire f1 by 1.0
#draw
intersect f1 with cyll as tbird3d
#draw
list
save tbird3d to tbird3d.sat

```

Note that the ACIS.mon file demonstrates how 3D solid models may be constructed starting from an initial FASTQ profile followed by typical solid modeling commands (e.g. sweep, intersect) resulting in the desired geometry.

In this example, only the 3D shell of the thunderbird is desired for the finite element model, and thus, the block command is used to specify that only elements on the surface are to be created. The following CUBIT journal file demonstrates current 3D paving capability:

1. The *fsqacs* users manual is reproduced in Appendix C, “Fsqacs: A FASTQ to ACIS Command Interpreter” on page 163

#File: tbird3d.jou

```

Import Acis 'tbird3d.sat'
#Display
View From 6 3 10
Label Surface on
Display
Draw Surface 23
Draw Surface 24

Surface 24 Size 0.03
Surface 24 Scheme Pave
Mesh Surface 24
Draw Surface 24

Block 1 Surface 24
Block 1 Attribute 0.03

```

▼ Assembly Components

Finally, a more practical example of ACIS/CUBIT is demonstrated by meshing an electronics assembly package. Figure B-7 shows a section of the assembly model containing three components: the accelerometer, the timer, and the radar. Also shown is the low density foam encapsulating these components. Note that the foam is of conical shape and the timer and radar units both have draft angles.

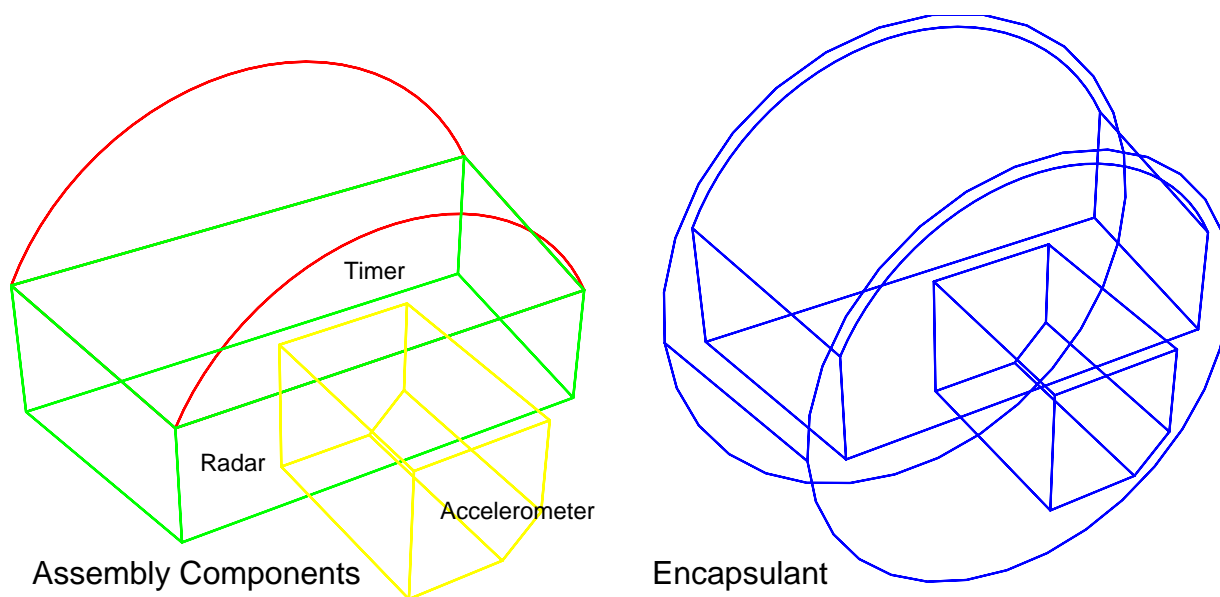


Figure B-7 Components in electronics assembly package.

In this case, the ACIS solid model is constructed on a component by component basis, and the final model called **accelLayer.sat** is generated by grouping the separate ACIS volumes together as one ACIS body. The user may prefer to create the entire solid model in a single ACIS session. However, for demonstration purposes, the model constructed here consists of five ACIS.mon files and one FASTQ input file that is converted to ACIS input using **fsqacs**. A

makefile is used to manage the input and output files and efficiently generate the model. The mesh generated for this assembly is shown in Figure B-8.

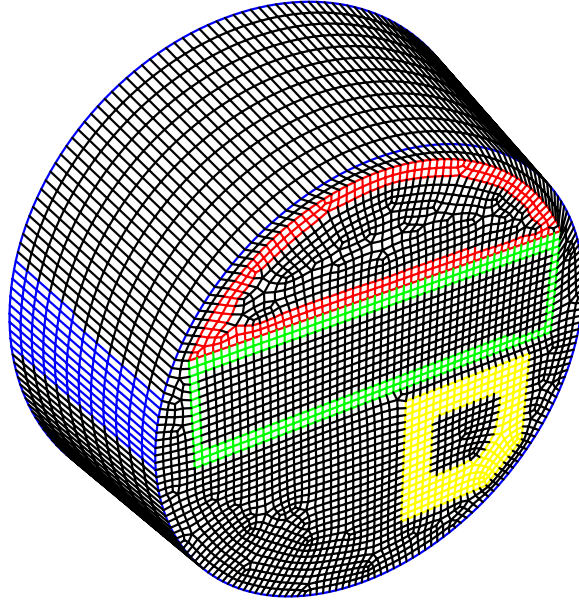


Figure B-8 Generated mesh for the electronics assembly package.

The complete geometric description is given by the following input files.

#File: timer.mon

```
option props on
cylinder cyll height 2.107 radius 2 top 2.362
view from 0 0 1 scale 50

block topBlock width 6 depth 6 height 6
move topBlock 0 3 3
rotate topBlock by -4.41 about x
move topBlock 0 .8976 -1.0535

intersect topBlock with cyll as timer
#draw
save timer to timer.sat
```

#File: radar.mon

```
option props on
cylinder cyll height 2.107 radius 2 top 2.362
block topBlock width 6 depth 6 height 6
move topBlock 0 3 3
rotate topBlock by -4.41 about x
move topBlock 0 .8976 -1.0535

block rightBlock width 6 depth 6 height 6
move rightBlock 3 0 3
rotate rightBlock by 8.734 about y
move rightBlock 0 0 -1.0535
move rightBlock 1.787 0 0
copy rightBlock as leftBlock
```

```

reflect leftBlock along x
unite rightBlock with leftBlock as sliceBlocks
#view from 0 0 1 scale 50
#draw

unite topBlock with sliceBlocks
#draw
subtract sliceBlocks from cyl1 as radar
#draw

block bottomBlock width 6 depth 6 height 6
move bottomBlock 0 -3.125 0

subtract bottomBlock from radar
#draw
save radar to radar.sat

```

#File: accel.mon

```

include accel.acs
view from 0 0 1 scale 50
#draw

sweep wire f1 by 2.107 direction 0 0 1
move f1 0 0 -1.0535
copy f1 as accel
save accel to accel.sat

```

#File: foam.mon

```

option props on
cylinder cyl1 height 2.107 radius 2 top 2.362
view from 0 0 1 scale 50

block rightBlock width 6 depth 6 height 6
move rightBlock 3 0 3
rotate rightBlock by 8.734 about y
move rightBlock 0 0 -1.0535
move rightBlock 1.787 0 0
copy rightBlock as leftBlock
reflect leftBlock along x
unite rightBlock with leftBlock as sliceBlocks
#draw

subtract sliceBlocks from cyl1 as hole
block bottomBlock width 6 depth 6 height 6
move bottomBlock 0 -3.125 0
subtract bottomBlock from hole
#draw hole

retrieve accel.sat as accel
unite accel with hole
#draw hole

cylinder foam height 2.107 radius 2.124 top 2.486
subtract hole from foam
#draw
save foam to foam.sat

```

#File: accelLayer.mon

```

option props on
view from 0 0 1 scale 50

retrieve timer.sat as timer
retrieve radar.sat as radar
retrieve accel.sat as accel
retrieve foam.sat as foam

```

```
#draw

group timer radar accel foam as accelLayer
#draw
save accelLayer to accelLayer.sat
```

ACIS commands worth noting in this example include:

- **option props on** Inserts an edge or “scribe line” along the outer surface of a cylinder. This changes the *periodic*¹ surface into a surface with only one bounding exterior loop of edges. Some CUBIT meshing algorithms require this type of solid model when constructing geometry using cylinders, spheres, or tori.
- **save** Individual components may be saved as separate ACIS solid models.
- **retrieve** Any valid ACIS.sat file may be retrieved and used to perform booleans and/or transformations in an ACIS session.
- **group** Individual components (ACIS bodies) may be grouped together to create a single ACIS.sat file for an assembly.

The resulting solid model is meshed in CUBIT using the following commands.

```
#File: accelLayer.jou
journal off
Import Acis 'accelLayer.sat'
Merge All
Display
View From 3 4 -5
Display
# front face of foam encapsulant
Surface 28 Size .07
Surface 28 Scheme Pave
Mesh Surface 28
# front face of accelerometer
Surface 3 Size .07
Surface 3 Scheme Pave
Mesh Surface 3
# front face of radar
Surface 7 Size .07
Surface 7 Scheme Pave
Mesh Surface 7
# front face of timer
Surface 16 Size .07
Surface 16 Scheme Pave
Mesh Surface 16
Display
# foam encapsulant
Volume 4 Interval 12
Volume 4 Scheme Project Source 28 Target 29
Mesh Volume 4
# accelerometer
Volume 3 Interval 12
Volume 3 Scheme Project Source 16 Target 17
Mesh Volume 3
# radar
Volume 2 Interval 12
Volume 2 Scheme Project Source 7 Target 10
Mesh Volume 2
Volume 2 Interval 12
# timer
Volume 1 Scheme Project Source 3 Target 4
```

1. A periodic surface is one which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction.

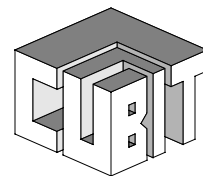
```
Mesh Volume 1
#Display
Block 1 Volume 1
Block 2 Volume 2
Block 3 Volume 3
Block 4 Volume 4
Export Genesis 'accelLayer.exoII'
```

This example demonstrates that setting the number of intervals for every edge in a 3D solid model can be a very tedious task. When possible, users should use geometry consolidation to reduce the amount of effort involved in performing this step. Additionally, clever use of the body interval command can also significantly reduce time and effort. In this example, all components have the same number of intervals in the z-direction. It is advantageous to set this value for all edges parallel to the z-axis by using the body interval command. Finally, when a mesh is projected from a source surface to a target surface, if one of the surfaces is larger than the other (i.e., if the swept region contains a draft angle), a better quality mesh will usually be generated if the smaller of the two surfaces is used as the source surface.

▼ Whisker Weaving

The following example demonstrates the ability to whisker weave a volume defined by the surface mesh, without any underlying ACIS geometry.

```
#File: ???jou
brick width 10
volume 1 interval 3
mesh surface 1 to 6
block 1 surface 1 to 6
export genesis "block3-3-3.gen"
display
pause
reset
import geometry mesh "block3-3-3.gen" block 1
display
pause
volume 1 scheme weave
set primal on
set query off
volume 1 smooth scheme laplacian
mesh volume 1
display
```



Appendix C: CUBIT Installation

▼ Licensing...	175
▼ Distribution Contents...	176
▼ Installation...	176
▼ HyperHelp Installation...	176

This Appendix contains information about the licensing and redistribution restrictions attached to CUBIT, the distribution contents, and installation instructions. All questions pertaining to obtaining a license for CUBIT should be directed to:

*Marilyn K. Smith
Technology Programs Department
Division 1503, MS-0833
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0833
Fax: (505) 844-9297, Email: mksmith@sandia.gov*

▼ Licensing

CUBIT is distributed in statically linked executable form for each supported platform. Supported platforms include the HP 9000 series running HP-UX¹, Sun SPARCstations running SunOS² and Solaris, and the SGI running IRIX³. Additional platforms will be added as required.

Note: CUBIT installations have use restrictions. THE CUBIT CODE CANNOT BE COPIED TO ANOTHER COMPUTER AND THE NUMBER OF USER SEATS ON EACH COMPUTER OR LAN IS LIMITED. If additional user seats or additional copies of CUBIT are required, you MUST contact us to acquire them.

CUBIT incorporates code modules developed by outside code vendors and our license agreements with them limit the number of user seats at Sandia National Laboratories and limit the number of users who are doing work in conjunction with Sandia National Laboratories.

-
1. HP-UX is a registered trademark of Hewlett-Packard Company.
 2. Sun, SunOS, and Solaris are registered trademarks of Sun Microsystems, Inc.
 3. IRIX is a registered trademark of Silicon Graphics, Inc.

Hence, CUBIT cannot be copied and redistributed without affecting the licensing agreement with the vendors who have proprietary interests in code modules within CUBIT.

Code distributions within Sandia National Laboratories are managed by an informal memorandum. Code distributions outside Sandia National Laboratories are managed by either a Use Notice memorandum or by a formal license agreement depending upon the code recipient. Use Notice and license agreement formats have been developed by the legal department at Sandia National Laboratories to protect the copyrights of code vendors and to protect the commercialization of Sandia National Laboratories copyrights to the CUBIT and SEACAS codes.

▼ Distribution Contents

In addition to the CUBIT executable, a code distribution can include example inputs and a test suite for CUBIT and, depending upon the nature of the request for CUBIT, a code distribution could include certain codes from the Sandia National Laboratories Engineering Analysis Code Access System [14] (SEACAS). Codes in SEACAS which could be used with CUBIT include finite element analysis codes, graphical postprocessing codes, and non-graphical pre- and postprocessing codes. Note that all codes, whether CUBIT or SEACAS codes, run under UNIX¹ operating systems.

Distributions containing other programs in addition to CUBIT will be supplied in tar format. For users who cannot access the tar file through ftp, the tar file will be written to magnetic or CD-ROM media and mailed. Due to possible exposure of the code and subsequent violation of copyrights and export control regulations, no electronic mailing of CUBIT or other codes is permitted.

▼ Installation

CUBIT and supporting CUBIT examples are installed simply by unpacking the tar file and moving the executables to their final directory. Examples and test problems for CUBIT include a README file which provides information needed to run the test problems and examples.

Any SEACAS code distributed with CUBIT will be in source code only. The compilation, linking, and installation of executables is managed by a very complete and extensive installation script. A complete set of installation procedures is provided with the SEACAS codes.

▼ HyperHelp Installation

CUBIT uses an online help system from Bristol Technology called HyperHelp. This online help system allow the viewing of this document and any supporting documentation online. An X Window system is required to run HyperHelp.

1. UNIX is a registered trademark of UNIX Systems Laboratories Inc.

The HyperHelp Viewer files distributed with CUBIT are shown in Table C-1, “HyperHelp Distribution Files,” on page 177.

Table C-1HyperHelp Distribution Files

Filenames	Description
gunzip.hp700.Z gunzip.sun4.Z gunzip.sgi.Z	gunzip utility
hp700R51.tar.gz hp700R52.tar.gz	HyperHelp for HP
sgiR41.tar.gz sgiR42.tar.gz	HyperHelp for SGI IRIX4 and IRIX5
sun4R41.tar.gz sun4R42.tar.gz	HyperHelp for Sun OS 4.1/Solaris 1.1
runtime.tar.gz	Printer Configuration Files for All Platforms

This guide describes how to install your copy of HyperHelp from the HyperHelp installation media. To install HyperHelp 4, you must copy the HyperHelp files from the installation media, set up the HyperHelp environment.

System Requirements

Although HyperHelp4.0 supports many platforms and operating systems, the hardware and software requirement as supplied by the CUBIT distribution are as follows.

Hardware Requirements

• **CPU**

HP 9000 Series 700/800 systems

Silicon Graphics systems

Sun SPARC systems

• **Disk Space**

HyperHelp Viewer: 13MB (includes sample files and printer configuration files)

• **Printer**

Postscript Level 1 and Level 2

PCL Level 4 and Level 5

Software Requirements

• **Operating System**

HP-UX 9.05

IRIX 4.0.5F or IRIX 5.0

SunOS 4.1/Solaris 1.1

SunOS 5.3/Solaris 2.3 (available shortly)

• *Windowing Environment*

X11R5/Motif1.2

X11R4/Motif1.1.4

OpenWindows 3.0

Copying HyperHelp Files

Identify the directory you want to install HyperHelp in, create that directory (if necessary), and **cd** to it.

For example, if you want to install HyperHelp in `/opt/help`, enter the following commands:

```
mkdir /opt/help
```

```
cd /opt/help
```

The HyperHelp installation procedure will create a `hyperhelp` subdirectory in the current working directory.

Uncompress and unzip the HyperHelp files with the following commands:

```
uncompress *.Z
```

```
./gunzip.arch *.gz
```

The *arch* represents your platform architecture (for example, `sun4` or `hp700`).

Unarchive each file that ends with the `.tar` extension as follows:

```
tar xpvf filename.tar
```

The following table shows the HyperHelp installation directory structure:

Directory	Description
<i>install_dir</i> /hyperhelp/bin	Contains the HyperHelp Viewer.
<i>install_dir</i> /hyperhelp/Xp	Contains Xprinter configuration files.
<i>install_dir</i> /hyperhelp/RELEASE	Release information text file.
<i>install_dir</i> /hyperhelp/hoh.hlp	How to Use HyperHelp help file.

Setting Up the HyperHelp Environment

Set the `HHHOME` environment variable to the location of the HyperHelp files.

C shell users: Add the following to your `.cshrc` or `.login` file:

```
setenv HHHOME install_dir/hyperhelp
```

Korn shell or Bourne shell users: Add the following to your `.profile` file:

```
HHHOME=install_dir/hyperhelp;export HHHOME
```

Add `$HHHOME/bin` to your `PATH` environment variable.

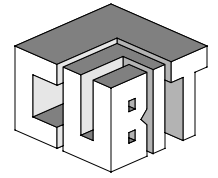
If you have an earlier version of HyperHelp installed on your system, make sure you add `$HHHOME/bin` BEFORE the old HyperHelp path in your `PATH` environment variable.

Activate your HyperHelp X resources with the following command:

cp \$HHHOME/app-defaults /usr/lib/X11/app-defaults/HyperHelp

If you are unable to get access to this directory, you can append the contents of \$HHHOME/app-defaults to \$HOME/.Xdefaults.

Log out and log back in to your system to restart your X server.



Appendix D: Available Colors

Table 6-2 in this Appendix lists the colors available in CUBIT at this time. All color commands require the specification of the color name. The table in this appendix lists the color number (#), color name, and the red, green, and blue components corresponding to each color for reference.

Table 6-2 Available Colors

#	Color Name	Red	Green	Blue
0	black	0.000	0.000	0.000
1	red	1.000	0.000	0.000
2	green	0.000	1.000	0.000
3	yellow	1.000	1.000	0.000
4	blue	0.000	0.000	1.000
5	magenta	1.000	0.000	1.000
6	cyan	0.000	1.000	1.000
7	white	1.000	1.000	1.000
8	grey	0.500	0.500	0.500
9	orange	1.000	0.647	0.000
10	pink	1.000	0.753	0.796
11	brown	0.647	0.165	0.165
12	gold	1.000	0.843	0.000
13	lightblue	0.678	0.847	0.902
14	lightgreen	0.000	0.800	0.000
15	salmon	0.980	0.502	0.447
16	coral	1.000	0.498	0.314
17	purple	0.627	0.125	0.941
18	paleturquoise	0.686	0.933	0.933

Table 6-2 Available Colors

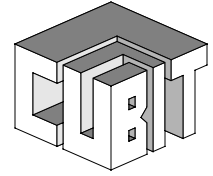
#	Color Name	Red	Green	Blue
19	lightsalmon	1.000	0.627	0.478
20	springgreen	0.000	1.000	0.498
21	slateblue	0.416	0.353	0.804
22	sienna	0.627	0.322	0.176
23	seagreen	0.180	0.545	0.341
24	deepskyblue	0.000	0.749	1.000
25	khaki	0.941	0.902	0.549
26	lightskyblue	0.529	0.808	0.980
27	turquoise	0.251	0.878	0.816
28	greenyellow	0.678	1.000	0.184
29	powderblue	0.690	0.878	0.902
30	mediumturquoise	0.282	0.820	0.800
31	skyblue	0.529	0.808	0.922
32	tomato	1.000	0.388	0.278
33	lightcyan	0.878	1.000	1.000
34	dodgerblue	0.118	0.565	1.000
35	aquamarine	0.498	1.000	0.831
36	lightgoldenrodyellow	0.980	0.980	0.824
37	darkgreen	0.000	0.392	0.000

Table 6-2 Available Colors

#	Color Name	Red	Green	Blue
38	lightcoral	0.941	0.502	0.502
39	mediumslateblue	0.482	0.408	0.933
40	lightseagreen	0.125	0.698	0.667
41	goldenrod	0.855	0.647	0.125
42	indianred	0.804	0.361	0.361
43	mediumspringgreen	0.000	0.980	0.604
44	darkturquoise	0.000	0.808	0.820
45	yellowgreen	0.604	0.804	0.196
46	chocolate	0.824	0.412	0.118
47	steelblue	0.275	0.510	0.706
48	burlywood	0.871	0.722	0.529
49	hotpink	1.000	0.412	0.706
50	saddlebrown	0.545	0.271	0.075
51	violet	0.933	0.510	0.933
52	tan	0.824	0.706	0.549
53	mediumseagreen	0.235	0.702	0.443
54	thistle	0.847	0.749	0.847
55	palegoldenrod	0.933	0.910	0.667
56	firebrick	0.698	0.133	0.133
57	palegreen	0.596	0.984	0.596
58	lightyellow	1.000	1.000	0.878
59	darksalmon	0.914	0.588	0.478

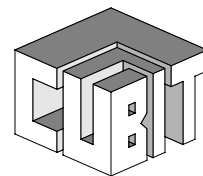
Table 6-2 Available Colors

#	Color Name	Red	Green	Blue
60	orangered	1.000	0.271	0.000
61	palevioletred	0.859	0.439	0.576
62	limegreen	0.196	0.804	0.196
63	mediumblue	0.000	0.000	0.804
64	blueviolet	0.541	0.169	0.886
65	deeppink	1.000	0.078	0.576
66	beige	0.961	0.961	0.863
67	royalblue	0.255	0.412	0.882
68	darkkhaki	0.741	0.718	0.420
69	lawngreen	0.486	0.988	0.000
70	lightgoldenrod	0.933	0.867	0.510
71	plum	0.867	0.627	0.867
72	sandybrown	0.957	0.643	0.376
73	lightslateblue	0.518	0.439	1.000
74	orchid	0.855	0.439	0.839
75	cadetblue	0.373	0.620	0.627
76	peru	0.804	0.522	0.247
77	olivedrab	0.420	0.557	0.137
78	mediumpurple	0.576	0.439	0.859
79	maroon	0.690	0.188	0.376
80	lightpink	1.000	0.714	0.757
81	darkslateblue	0.282	0.239	0.545
82	rosybrown	0.737	0.561	0.561
83	mediumvioletred	0.780	0.082	0.522
84	lightsteelblue	0.690	0.769	0.871
85	mediumaquamarine	0.400	0.804	0.667



References

- 1 T. D. Blacker and M. B. Stephenson, 'Paving: a new approach to automated quadrilateral mesh generation', SAND90-0249, Sandia National Laboratories, (1990).
- 2 M. B. Stephenson, S. A. Canann, and T. D. Blacker, 'Plastering: a new approach to automated, 3D hexahedral mesh generation', SAND89-2192, Sandia National Laboratories, (1992).
- 3 G. D. Sjaardema, et. al., *CUBIT Mesh Generation Environment, Volume 2: Developers Manual*, SAND94-1101, Sandia National Laboratories, (1994).
- 4 Spatial Technology, Inc., *ACIS Test Harness Application Guide Version 1.4*, Spatial Technology, Inc., Applied Geometry, Inc., and Three-Space, Ltd., (1992).
- 5 T. D. Blacker, *FASTQ Users Manual Version 1.2*, SAND88-1326, Sandia National Laboratories, (1988).
- 6 L. A. Schoof, *EXODUS II Application Programming Interface*, internal memo, Sandia National Laboratories, (1992).
- 7 W. A. Cook and W. R. Oakes, 'Mapping methods for generating three-dimensional meshes', *Comp. mech. eng.*, **Volume 1**, 67-72 (1982).
- 8 R. E. Jones, *QMESH: A Self-Organizing Mesh Generation Program*, SLA - 73 - 1088, Sandia National Laboratories, (1974).
- 9 R. E. Tipton, 'Grid Optimization by Equipotential Relaxation', unpublished, Lawrence Livermore National Laboratory, (1990).
- 10 A. P. Gilkey and G. D. Sjaardema, *GEN3D: A GENESIS Database 2D to 3D Transformation Program*, SAND89-0485, Sandia National Laboratories, (1989).
- 11 G. D. Sjaardema, *GREPOS: A GENESIS Database Repositioning Program*, SAND90-0566, Sandia National Laboratories, (1990).
- 12 G. D. Sjaardema, *GJOIN: A Program for Merging Two or More GENESIS Databases*, SAND92-2290, Sandia National Laboratories, (1992).
- 13 G. D. Sjaardema, *APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses*, SAND92-2291, Sandia National Laboratories, (1992).
- 14 G. D. Sjaardema, *Overview of the Sandia National Laboratories Engineering Analysis Code Access System*, SAND92-2292, Sandia National Laboratories, (1993).
- 15 S. C. Lovejoy and R. G. Whirley, *DYNA3D Example Problem Manual*, UCRL-MA--105259, University Of California and Lawrence Livermore National Laboratory, (1990).
- 16 Open Software Foundation, Inc., *OSF/Motif™ User's Guide Revision 1.2*, PTR Prentice Hall, Englewood Cliffs, New Jersey, (1993).
- 17 J. M. Osier, *Keeping Track, Managing Messages with GNATS, The GNU Problem Report Management System*, Users manual for GNATS Version 3.2, Cygnus Support, October 1993.
- 18 L. M. Taylor and D. P. Flanagan, *Pronto 3D—A Three-Dimensional Transient Solid Dynamics Program*, SAND87-1912, Sandia National Laboratories, (1989).
- 19 S. W. Attaway, unpublished, (1993).



Glossary

B

Body. A body is simply a collection or set of volumes. It differs from volumes only in the fact that booleans are only performed between bodies, not between volumes. The simplest body contains one volume. 64

Brick. A brick is a hexahedral element defined by six connected faces. A brick is owned by the enclosing volume. 82

C

Curve. A curve is a line (not necessarily straight) which is bounded by at least one but not more than two vertices. 64

E

Edge. An edge is defined by a minimum of two nodes. Additional nodes may exist on the edges of higher-order elements. An edge on a curve is owned by that curve, an edge in a surface is owned by that surface, and an edge in a volume is owned by that volume 82

Element Blocks. Element Blocks (also referred to as simply, Blocks) are a logical grouping of elements all having the same basic geometry and number of nodes. 133

F

Face. A face is defined by four connected edges. A face on a surface is owned by that surface, a face in the interior of of a volume is owned by that volume. 82

G

Geometry primitives. Classes of general geometric shapes which are differentiated by basic parameters. CUBIT supports the brick, pyramid, prism, cylinder, torus, frustum, and sphere. 66

H

Hard Point. A vertex which is located in the interior of a surface. It is used to force a node location to that specific geometric location. 64

N

Node A node is a single point in space. A node at a vertex is owned by that geometric vertex, a node on a curve is owned by that curve, a node on the interior of a surface is owned by that surface, and a node in a volume is owned by that volume. 81

Nodeset. Nodesets are a logical grouping of nodes also accessed through a single ID known as the Nodeset ID. 133

P

Periodic Surface. A periodic surface is a surface which is not contained within a single exterior loop of edges. It is termed periodic because the regular parameterization of the surface will have a jump from 0 to 2π in the periodic direction. 64

S

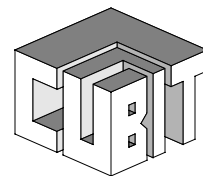
Sideset. Sidesets represent a grouping of element sides and are also referenced using an integer Sideset ID. 134

Surface. A surface in CUBIT is a finite bounded portion of some geometric surface (finite or infinite). A set of surfaces bound the volume in a volume. A surface is bounded by a set of curves. 64

V

Vertex. A vertex occupies a single point in space. A vertex is used to bound a curve and/or to specify a specific location for a node. 64

Volume. Volumes are volumetric regions and are always bounded by one or more surfaces. For practical consideration, volumes will always be bounded by two or more surfaces. 64



Index

Symbols

\$HOME/.cubit 19, 20
.cubit 19, 20
.Xdefaults 21
.Xresources 21

Numerics

2-manifold topology 65

A

ACIS 24, 147, 159
 Test Harness 24, 71, 77, 159, 169
Active
 Window 43
adaptivity 25
airfoil 163
Angle
 Perspective 45, 47, 146
Animation 155
 Pan 146, 150
 Rotate 151
 View 139, 145, 154, 155
 Zoom 146, 157
Aprepro 167
Aries® ConceptStation 24, 77, 159
aspect ratios 164
Assembly Components 170
At 45, 47, 139, 155
Attribute 134
 Block 134, 139
Autocenter 44, 145
Autoclear 44, 48, 145
Axis 44, 145

B

Background Color 43, 50, 141
-batch 19, 20
Bias 88, 143
 Reverse 143
Block 162, 166
 Attribute 134, 139
 Color 50, 141
 Curve 134
 Draw 144
 Element Type 134, 139
 Geometry Color 50
 Geometry Type 139
 Mesh Color 50
 Surface 134
 Visibility 140
 Volume 134
Body 64
 Color 50, 141
 Copy 72, 140
 Decomposition 156
 Draw 48, 144
 Geometry Color 50, 141
 Geometry Visibility 140
 Interval 88, 140, 174
 Interval Size 88, 140
 Label 51, 148
 List 52, 54, 148
 Mesh 149
 Mesh Color 50, 141
 Mesh Visibility 140
 Move 72, 140
 Reflect 73, 140
 Restore 73, 140
 Rotate 72, 140
 Scale 72, 140
 Visibility 50, 140
Webcut 156

Booleans 73
 Intersect 73, 147, 161
 Subtract 73, 152, 160, 161
 Unite 73, 154
 Border 44, 145
 Boundary Condition 82, 134
 Contact Surface 134
 NodeSet 133, 136
 SideSet 134, 136
 BoundaryLayer 103, 105, 163
 Curve 141
 Parameters 141
 Surface 141
 Box Beam 164
 Brick 66, 141, 142, 160, 161

C

Cellular Topology 65
 Center 44, 45, 145
 Clear 45, 48, 145
 Color 50
 Background 43, 50, 141
 Block 50, 141
 Body 50, 141
 Geometry 141
 Mesh 141
 Geometry 141
 Menu 50
 Mesh
 Surface 142
 Volume 142
 Node 50, 141
 NodeSet 50
 Nodeset 141
 SideSet 50, 141
 Surface 50, 142
 Geometry 142
 Mesh 142
 Table 181
 Volume 50, 142
 Geometry 142
 Mesh 142
 Command Line
 Echo 41, 145
 Interface 39

Constraints Menu 136
 Contact Surface 134
 Copy 161
 Body 72, 140
 Mesh 128, 142
 Create 66
 Brick 66, 142
 Cylinder 66, 67, 143
 Dialog Box 66
 Frustum 66, 68, 142, 145
 Prism 66, 68, 142, 151
 Pyramid 66, 68, 143, 151
 Sphere 66, 69, 143, 152
 Torus 66, 69, 143, 154
 Window 43
 Cube with Hole 28, 160
 cubit 19
 CUBIT_HELP_DIR 21
 CUBIT_OPT 20, 21
 cubitb 19, 40
 cubitHelpGUI.hlp 21
 cubitx 19, 21
 Cursor
 Pan 146, 150
 Zoom 47, 157
 Curvature
 Curve Scheme 143
 Surface Scheme 153
 Volume Scheme 156
 Curve 64, 135
 Bias 143, 147, 163
 Block 134
 BoundaryLayer 105
 Curvature 143
 Delete Mesh 126, 144
 Draw 48, 144
 Equal 143
 Interval 88, 143, 147
 Interval Size 88, 143, 147
 Label 51, 148
 List 52, 54, 148
 Merge 79
 Mesh 90, 149
 NodeSet 136, 150
 Reverse Bias 89, 143, 147
 Scheme

- Bias 88, 143, 147
- Curvature 143
- Equal 88, 143
- SideSet 136, 152
- Type 143
- Cylinder 66, 67, 142, 143, 160, 161

D

- Debug 58
 - List 148
 - Set 151
- debug 20, 58
- Decompose 143
- Decomposition 75, 156
- Delete
 - Face 127, 144
 - Mesh 126, 144
 - Window 43
- DISPLAY 20
- Display 48, 144, 150
- Draw 48
 - Block 144
 - Body 48, 144
 - Curve 48, 144
 - Edge 48, 144
 - Face 48, 144
 - Hex 48, 144
 - Loop 144
 - Node 48, 144
 - NodeSet 48, 144
 - SideSet 48, 145
 - Skeleton 145
 - Surface 48, 145
 - Vertex 48, 145
 - Volume 48, 145
 - Mesh 155

E

- Echo 41, 145
 - List 148
 - Set 151
- Edge
 - Draw 48, 144

- Label 51, 148
- Editing
 - Mesh 124
- Element Block 25, 133
- Element Type 87, 139
 - Block 134
- Encapsulated 147
- Environment Variable
 - CUBIT_HELP_DIR 21
 - CUBIT_OPT 20, 21
 - DISPLAY 20
 - HOME 20
 - PATH 20
- EPS 51, 147
- Equal 88, 143
- Equipotential 125
 - Area 126
- Error 58
- Example
 - Assembly Components 170
 - Box Beam 164
 - Cube with Hole 28, 160
 - Octant of Sphere 161
 - Thunderbird 3D Shell 167
- Execution Options
 - batch 19, 20
 - debug 20, 58
 - fastq 20
 - help 19
 - Include 20
 - information 20, 58
 - initfile 19, 20
 - input 20
 - maxjournal 20
 - noinitfile 19
 - nojournal 19, 20, 21, 42
 - solidmodel 19
 - warning 20, 58
- Exit 41, 145
- Exodus 133
- ExodusII 127
- Export
 - Genesis 137, 145

F

Face

- Delete 127, 144
- Draw 48, 144
- Label 51, 148
- List 52, 54, 148

False (toggle) 22

FASTQ 24, 70, 147

- Import 70

-fastq 20

Filename 22

Files

- \$HOME/.cubit 19, 20
- .Xdefaults 21
- .Xresources 21
- cubitHelpGUI.hlp 21
- Exodus 133
- ExodusII 127
- Genesis 133, 137, 145

FlatShade 44, 146

From 45, 46, 47, 145, 155

Frustum 66, 68, 142, 145

fsqacs 159, 169

G

Genesis 133, 137, 145

- Export 137

Geometry

- Booleans 73
- Color 141
 - Body 50
 - NodeSet 50
 - SideSet 50
 - Surface 50, 142
 - Volume 50, 142

Creation 66

Decomposition 75

Definition 63

Label 51, 148

Manipulation 71

Menu 66

Merge 24, 50, 161, 166, 174

Primitives 66

Type 139

Visibility 49, 140, 145

- Surface 153

- Volume 155

Graphics

- Autocenter 44, 145

- Autoclear 44, 48, 145

- Axis 44, 145

- Border 44, 145

- Center 45, 145

- Clear 45, 48, 145

- Display 48

- Draw 48

- Line Width 45, 146

- List

- View 47

Mode

- FlatShade 44, 146

- HiddenLine 43, 146

- Painters 44, 146

- PolygonFill 44, 146

- SmoothShade 44, 146

- WireFrame 43, 146

Mode Type 43

Pan 146, 150

Perspective 47, 146

- Angle 45, 47, 146

Rotate 46

Text Size 51

Window

- Active 43

- Create 43

- Delete 43

Window Create 43

Window Size 43, 146

Zoom 47, 146, 157

- Cursor 47

- Reset 47

- Screen 47

GUI 39

H

Hard point 64

Hard Set 86, 88

Hardcopy 51, 147

Hardware Platforms 25

Help 61, 147
 Hyperhelp 61, 147
 -help 19
 Hex
 Draw 48, 144
 Label 51, 148
 List 52, 54, 148
 Weight 126, 156
 Weight Surface 126, 156
 Weighting Function 126
 HiddenLine 43, 146
 HOME 20
 Hyperhelp 61, 147

I

Import
 Acis 147
 Fastq 70, 147
 Mesh 103, 128, 147
 -Include 20
 Information 58
 List 148
 Set 151
 -information 20, 58
 -initfile 19, 20
 Initialization File 19
 Initialize
 Video 51, 155
 -input 20
 Intersect 73, 147, 161
 Interval
 Body 88, 140
 Curve 88, 143, 147
 Default 82
 Hard Set 86, 88
 Size
 Body 88, 140
 Curve 88, 143, 147
 Surface 88, 154
 Volume 88, 156
 Surface 88, 153
 Volume 88, 155

J

Journal
 List 148
 Pause 42, 150
 Playback 42, 150
 Record 42, 151
 Set 151
 Journal Off 19, 20, 42, 148
 -journalfile
 Execution Options
 -journalfile 19

L

Label 50
 All 51, 148
 Body 51, 148
 Curve 51, 148
 Edge 51, 148
 Face 51, 148
 Geometry 51, 148
 Hex 51, 148
 Mesh 51, 148
 Node 51, 148
 Surface 51, 148
 Vertex 51, 148
 Volume 51, 148
 Laplacian 125
 Length-weighted Laplacian 125
 Line Width 45, 146
 List 52
 Body 52, 54, 148
 Curve 52, 54, 148
 Debug 148
 Echo 148
 Face 52, 54, 148
 Hex 52, 54, 148
 Information 148
 Journal 148
 Model 148
 Node 148
 Nodes 52, 54
 Settings 58, 148
 Surface 52, 54, 148
 Totals 148

- Vertex 52, 54, 148
- View 47, 48, 148, 155
- Volume 52, 54, 148
- Warning 148
- List Debug 148
- Loop 109, 144
 - Draw 144

M

- makefile 167
- Manifold model 65
- Map 94
 - Scheme
 - Surface 153
 - Volume 106, 156
- maxjournal 20
- Menu
 - Color 50
 - Constraints 136
 - Geometry 66
 - Graphics Mode Type 43
 - View 45
 - Visibility 48
- Merge 24, 50, 161, 166, 174
 - All 78, 149
 - Curve 79
 - General 78
 - Only Curves 79
 - Only Surfaces 79
- Mesh
 - Body 149
 - Color 141
 - Block 50
 - Body 50
 - NodeSet 50
 - SideSet 50
 - Surface 50, 142
 - Volume 50
 - Copy 128, 142
 - Curve 90, 149
 - Delete 126, 144
 - Editing 124
 - Import 103, 128, 147
 - Label 51, 148
 - Modify Smooth 124

- Smooth
 - Modify 124
- Surface 105, 149
- Visibility 49, 140, 149
 - Surface 153
- Volume 105, 124, 149
 - Visibility 155

Messages

- Debug 58
- Error 58
- Information 58
- Warning 58

Model

- attributes 24
- List 148

Move

- Body 72, 140
- NodeSet 127, 150

N

- No (toggle) 22

Node

- Color 50, 141
- Draw 48, 144
- Label 51, 148
- List 52, 54, 148
- Repositioning 127
- Visibility 49, 149
- NodeSet 25, 133, 136
 - Color 50, 141
 - Curve 136, 150
 - Draw 48, 144
 - Geometry Color 50
 - Mesh Color 50
 - Move 127, 166
 - Move To 150
 - Surface 136, 150
 - Vertex 136, 150
 - Visibility 49, 150, 152
 - Volume 136, 150

- noinitfile 19

- nojournall 19, 20, 21, 42

- Non-manifold topology 65

O

Octant of Sphere 161
 Off (toggle) 22
 On (toggle) 22
 option props on 173
 Output
 PICT 51
 PostScript 51

P

Painters 44, 146
 Pan 150
 Cursor 146
 Parameter 22
 Optional 23
 PATH 20
 Pause 42, 150
 Pave 23, 94, 160, 161, 163
 Surface Scheme 153
 Perspective 47, 146
 Angle 45, 47, 146
 PICT 51
 Plaster 25
 Volume Scheme 106, 111, 156
 Playback 42, 150
 Plot 150
 PolygonFill 44, 146
 PostScript 51, 147
 PostScript Begin 25
 PostScript End 51
 Primitives 66
 Brick 66, 141, 142
 Cylinder 66, 67, 142, 143
 Dialog Box 66
 Frustum 66, 68, 142, 145
 Geometry 66
 Prism 66, 68, 142, 151
 Pyramid 66, 68, 143, 151
 Sphere 66, 69, 143, 152
 Torus 66, 69, 143, 154
 Prism 66, 68, 142, 151
 PRO/Engineer 24, 71, 77, 159
 Project 161
 Volume Scheme 106, 108, 109, 156

Pyramid 66, 68, 143, 151

Q

Quit 41, 145, 151

R

Record 42, 151
 Stop 42, 151
 Reflect
 Body 73, 140
 Repositioning
 Node 127
 Reset 41, 151
 View 155
 Zoom 47, 157
 Restore
 Body 73, 140
 Reverse Bias 89, 143
 Rotate 46, 151, 161
 Body 72, 140
 Continuous 46
 Volume Scheme 106, 108, 111, 156

S

Scale
 Body 72, 140
 Scheme 82
 Bias 88
 Curvature 143, 153
 Designation 105
 Equal 88
 Map 94
 Surface 153
 Volume 106
 Pave 94, 153
 Plaster 106, 111
 Project 106, 108, 109
 Rotate 106, 108, 111
 Sweep 108
 Translate 106, 108, 111
 Triangle 94, 96, 153
 Volume 106

- Curvature 156
- Map 156
- Plaster 156
- Project 156
- Rotate 156
- Translate 156
- Weave 156
- Weave 106, 113
- Screen
 - Zoom 157
- Set
 - Debug 151
 - Echo 151
 - Information 151
 - Journal 151
 - Warning 151
- Settings
 - List 58, 148
- SideSet 25, 134, 136
 - Color 50, 141
 - Curve 136, 152
 - Draw 48, 145
 - Geometry Color 50
 - Mesh Color 50
 - Surface 136, 152
 - Visibility 49, 152
- Size
 - Body 140
 - Curve 143, 147
 - Surface 154
 - Volume 156
- Skeleton
 - Draw 145
- Smooth
 - Equipotential 125
 - Equipotential Area 126
 - Equipotential Fixed 125
 - Equipotential Generic 126
 - Equipotential Inverse Area 125, 126
 - Equipotential Jacobian 125
 - Laplacian 125
 - Length-weighted Laplacian 125
 - Modify Mesh 124
 - Scheme 154, 156
 - Surface 124, 125, 152
 - Volume 125, 152
- Weight
 - Area 125
 - Inverse Area 125
 - Jacobian 125
- SmoothShade 44, 146
- Snap
 - Video 51, 155
- solidmodel 19
- Source Surface 106, 174
- Sphere 66, 69, 143, 152, 161
- String 22
- Subtract 73, 152, 160, 161
- Surface 64, 135, 153
 - Block 134
 - BoundaryLayer 105
 - Color 50, 142
 - Copy
 - Mesh 128
 - Curvature 153
 - Delete Mesh 126, 144
 - Draw 48, 145
 - Geometry Color 50, 142
 - Interval 88, 153
 - Interval Size 88, 154
 - Label 51, 148
 - List 52, 54, 148
 - Mapping 94
 - Merge 79
 - Mesh 105, 149
 - Visibility 153, 154
 - Mesh Color 50, 142
 - NodeSet 136, 150
- Scheme
 - Curvature 153
 - Map 94, 153
 - Pave 94, 153
 - Triangle 94, 153
- SideSet 136, 152
- Smooth 124, 152
 - Equipotential 125
 - Equipotential Area 126
 - Equipotential Fixed 125
 - Equipotential Generic 126
 - Equipotential Inverse Area 126
 - Equipotential Jacobian 126
 - Scheme 154

- Weight
 - Area 125
 - Inverse Area 125
 - Jacobian 125
- Source 106
- Target 106
- Visibility 50, 154
- Weight Hexes 126, 156
- Sweep
 - Volume Scheme 108

T

- Target Surface 106
- Test Harness 24
- Text Size 51
- Thunderbird 3D Shell 167
- Title 137, 154
- Toggle 22
- Topology
 - 2-manifold 65
 - Cellular 65
 - Non-manifold 65
- Torus 66, 69, 143, 154
- Totals
 - List 148
- Translate 160
 - Volume Scheme 106, 108, 111, 156
- Triangle 94, 96
 - Surface Scheme 153
- True (toggle) 22

U

- Unite 73, 154
- Up 45, 46, 47, 154, 155
- User interface 39

V

- Version 41, 154
- Vertex 64, 154
 - Delete Mesh 126, 144
 - Draw 48, 145
 - Label 51, 148

- List 52, 54, 148
- NodeSet 136, 150
- Visibility 49, 155
- Video 51
 - Initialize 51, 155
 - Snap 51, 155
- View
 - At 45, 47, 139, 155
 - Autocenter 44
 - Autoclear 44, 48
 - Border 44
 - Clear 48
 - FlatShade 44
 - From 45, 46, 47, 145, 155
 - HiddenLine 43
 - List 47, 48, 148, 155
 - Menu 45
 - Painters 44
 - Perspective
 - Angle 47
 - PolygonFill 44
 - Reset 155
 - SmoothShade 44
 - Up 45, 46, 47, 154, 155
 - Window Size 43
 - WireFrame 43
- Visibility
 - Block 140
 - Body 50, 140
 - Body Mesh 140
 - Geometry 49, 140, 145
 - Volume 155
 - Menu 48
 - Mesh 49, 149
 - Surface 153, 154
 - Node 49, 149
 - NodeSet 49, 150, 152
 - SideSet 49, 152
 - Surface 50
 - Geometry 153
 - Mesh 153, 154
 - Vertex 49, 155
 - Volume 50, 156
 - Mesh 155
- Volume 64, 135
 - Block 134

- Color 50, 142
- Copy
 - Mesh 128
- Delete Mesh 126, 144
- Draw 48, 145
- Geometry
 - Color 142
 - Visibility 155
- Geometry Color 50
- Interval 88, 155
 - Size 156
- Interval Size 88
- Label 51, 148
- List 52, 54, 148
- Map 106
- Mesh 124, 149
 - Color 142
 - Draw 155
 - Visibility 155
- Mesh Color 50
- Meshing 105
- NodeSet 136, 150
- Scheme 106
 - Curvature 156
 - Map 106, 156
 - Plaster 106, 111, 156
 - Project 106, 108, 109, 156
 - Rotate 106, 108, 111, 156
 - Sweep 108
 - Translate 106, 108, 111, 156
 - Weave 106, 113, 156
- Smooth 125, 152
 - Equipotential 125
 - Equipotential Fixed 125
 - Laplacian 125
 - Scheme 156
- Visibility 50, 156

W

- Warning 58
 - List 148
 - Set 151
- warning 20, 58
- Weave
 - Volume Scheme 106, 113, 156

- Webcut
 - Body 156
- Weight 125
- Weight Hexes
 - Surface 126, 156
- Weighting Function
 - Hex 126
- Window
 - Active 43
 - Create 43
 - Delete 43
- Window Create 43
- Window Size 43, 146
- WireFrame 43, 146

Y

- Yes (toggle) 22

Z

- Zoom 47, 146, 157
 - Cursor 47, 157
 - Reset 47, 157
 - Screen 47, 157

